Exercício Programa III : Modelagem de um Sistema de Resfriamento de Chips

MAP3121 - Métodos Numéricos e Aplicações

Prof. Dr. Pedro Peixoto

13 de julho, 2022

Laura do Prado Gonçalves Pinto
nºUSP 11819960

Mateus Stano Junqueira
nºUSP 11804845

Lista de Figuras

Conteúdo

1 Introdução

Neste relatório é apresentado todo o processo de construção lógica e computacional da resolução do terceiro exercício programa proposto para a disciplina MAP3121- Métodos Numéricos e Aplicações assim como os resultados obtidos pelos métodos apresentados e as conclusões tomadas sobre a tarefa proposta.

No exercício proposto trabalha-se a resolução e análise da função de calor de um chip e um resfriador, utilizando o método de elementos finitos. Para realizar esta tarefa, utilizam-se os códigos dos dois últimos exercícios programas a fim de fazer proveito do método de integração gaussiana para resolução de produtos internos e do método de solução de matrizes LU para resolver o sistema obtido no método de elementos finitos. Em seguida constrói-se um código para tecer análises sobre o comportamento de aquecimento e resfriamento do chip , dependendo das variações impostas à função de calor dada, incluindo variações de material, do comportamento constante e não constante dos calores gerado e absorvido, de condições de fronteira não homogêneas dentre outros descritos ao longo deste relatório, junto de suas respectivas conclusões.

Este relatório consta a metodologia utilizada para se alcançar tais objetivos além de todos os recursos utilizados para a construção do algoritmo assim como as análises referentes sobre os resultados encontrados.

2 Descrição

2.1 Análise do problema

O problema proposto na tarefa deste terceiro exercício programa consiste em utilizar o método de elementos finitos para resolução da equação de troca de calor referente à um chip e seu resfriador, respeitando as condições de contorno propostas e as devidas hipóteses simplificadoras. As aplicações consistem em validar o método utilizando-se dos códigos criados durante o primeiro e segundo exercícios programas deste curso resolvendo a equação principal para condições específicas as quais se tem respostas exatas e sabe-se a ordem de convergência. Em seguida, deve-se tratar do caso de um chip com material homogêneo e calcular o equilíbrio entre as forças de geração de calor do mesmo e a absorção de calor do resfriador, resolvendo as equações pelo método de elementos implementado em código para diferentes condições de contorno - assumindo primeiro funções de troca constantes e então adicionando-lhes complexidade. Então, tecem-se análises para um chip de material não constante, ou seja, seu k varia em função de x, e então utilizam-se os métodos para não somente chegar em soluções porém avaliar como diferentes materiais interferem nelas.

2.2 Equação do calor

O objetivo principal deste exercício programa , como mencionado, é uma modelagem térmica de um chip de tamanho L x L com espessura h. Para isso são adotadas inicialmente as seguintes hipóteses simplificadoras:

- Tratar o caso como unidimensional, analisando a seção transversal variando em x por todo o comprimento L .
- Assumindo que a espessura é suficientemente fina, pode-se assumir que não variação de temperatura na vertical.
- Ocorre apenas troca de calor na parte superior do chip, ou seja, a parte inferior é uma superfície perfeitamente isolada.

Utilizando a equação de Fourrier e o conceito de conservação de energia pode-se chegar na seguinte equação para delimitar a troca de calor entre o chip e seu resfriador :

$$\rho C \frac{\partial T(t,x)}{\partial t} = \frac{\partial (k(x) \cdot \frac{\partial T(t,x)}{\partial x})}{\partial x} + Q(t,x) \tag{1}$$

Na qual ρ é a densidade do material do chip; **C** é a constante de calor específico do material; **T(t,x)** é a função temperatura do chip para x no instante t; **k(x)** é a condutividade térmica do material na posição x; e **Q(t,x)** é a função da fonte de calor.

Quanto à função da fonte de calor , ela é o resultado da soma entre a função Q_+ gerado pelo chip e a função Q_- retirado pelo resfriador.

$$Q = Q_+ - Q_- \tag{2}$$

O valor de Q_+ pode ser obtido em função da potência **P** e do volume **V**:

$$Q_+ = \frac{P}{V} \tag{3}$$

Para resolver estas equações , precisa-se do estado inicial [T(0,x)] e dos estados de fronteira , ou seja valores de T para quando $x = 0$ e $x = L$.

Para esse modelo em específico adota-se que a temperatura nos extremos é igual à ambiente (20ºC).

2.3 Estado Estacionário

Para a hipótese de que o processador em questão está em regime constante, ou seja, Q_+ e Q_- são constantes tem-se que:

$$\frac{\partial T(t,x)}{\partial t} = 0 \longrightarrow \frac{-\partial(k(x)\frac{\partial T(x)}{\partial x})}{\partial x} = Q(x) \tag{4}$$

De tal forma que se Q_+ e Q_- forem conhecidos , assim como as condições de fronteira (**T(x=0)** e **T(x=L)**) pode-se encontrar soluções de equilíbrio com métodos numéricos para a equação do calor.

2.4 Método dos Elementos Finitos

Neste exercício programa, o método utilizado para resolução numérica das equações mencionadas será o método de elementos finitos.

Para a equação:

$$L(u(x)) := (-k(x)u'(x))' + q(x)u(x) = f(x)x \in (0,1), u(0) = u(1) = 0 \tag{5}$$

A solução clássica é dada por:

$$u(x) \in V_0 = v \in C^2[0,1] : v(0) = v(1) = 0 \tag{6}$$

Tem-se que, sendo $u(x)$ solução da equação (1) e $v(0) \in V_0$:

$$\int_0^1 L(u(x))v(x)\,dx = \int_0^1 f(x)v(x)\,dx \tag{7}$$

Utilizando o método de integração por partes e considerando que $u(x)$ e $v(x)$ se anulam nos extremos:

$$\int_0^1 [k(x)u'(x)v'(x) + q(x)u(x)v(x)]\,dx = \int_0^1 f(x)v(x)\,dx; \forall v \in V_0 \tag{8}$$

Caso $u(x) \in V_0$ satisfaça a equação (8), então u(x) é necessariamente solução de (5). Assim , para u(x) ambas as equações são equivalentes. Porém, ao passo que para (5) a solução precisa ser duas vezes continuamente diferenciável, para (8) pode-se formulá-la para equações mais gerais. De tal forma que é possível definir $u(x)$ e $v(x)$ em um espaço U_0 das funções contínuas, continuamente diferenciáveis por partes (com derivadas limitadas) e que se anulam nos extremos de [0, 1]. Tal que (8) também vale para $\forall v \in U_0$, chamada "versão fraca"de (5).

Nota-se que o lado esquerdo da equação (8) trata-se de um produto interno $\langle u, v\rangle_L \in U_0$. Para tal, imediatamente todas as propriedades de produtos internos valem para a expressão, assim para que o produto $u, u\rangle$ seja igual a zero, implica-se que $u = 0$ para quando $q = 0$. Como $u'(x)$ é necessariamente igual a zero no intervalo [0,1], tem-se que u é constante. Contudo, como nos extremos de u vale zero, tem-se que $u(x)$ é na verdade a função nula.

Para solucionar a equação (8) para todo v em U_0 , é possível utilizar o método de Ritz-Raleigh, o qual fará uma aproximação utilizando o método dos mínimos quadrados em um subespaço finito de U_0, U_n, através da projeção ortogonal de $u(x)$ neste subespaço.

Essa projeção é viável substituindo o produto interno usual pelo apresentado anteriormente. Assim a projeção ortogonal $\bar{u}_n$ de $u(x)$ em U_n é tal que $\langle u - \bar{u}_n, v_n \rangle = 0$ para todo v_n e u_n ou seja $\bar{u}_n, v_n\rangle = \langle u, v_n\rangle$.

Sendo u solução da equação e U_n está contido em U_0 tem-se que $v_n \in U_n, \langle u, v_n\rangle_L = \langle f, v_n\rangle$. É possível desta forma obter a projeção u por meio de $\bar{u}_n$: $\langle \bar{u}_n, v_n\rangle_L = \langle f, v_n\rangle \, , \forall v_n \in U_n$.

Como a função $\bar{u}_n$ minimiza o valor de $\|u - v_n\|_L = \langle u - v_n, u - v_n\rangle_L^{1/2}$, para $v_n \in U_n$, ela é a melhor aproximação da solução u no espaço U_n.

Para obter-se a solução da projeção mencionada, é necessário definir uma base $\phi_1 ... \phi_n$ do subespaço U_n. Após escolhida, e escrevendo $\bar{u}_n = \sum_{i=1}^{n} \alpha_i \phi_i$ basta resolver $\langle \bar{u}_n, \phi_i\rangle_L = \langle f, \phi_i\rangle \, , i = 1, \ldots, n$ no seguinte sistema linear:

$$\begin{bmatrix} \langle \phi_1, \phi_1\rangle_L & \langle \phi_2, \phi_1\rangle_L & \ldots & \langle \phi_n, \phi_1\rangle_L \\ \ldots & \ldots & \ldots & \ldots \\ \langle \phi_1, \phi_n\rangle_L & \langle \phi_2, \phi_n\rangle_L & \ldots & \langle \phi_n, \phi_n\rangle_L \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ .. \\ \alpha_n \end{bmatrix} = \begin{bmatrix} \langle f, \phi_1\rangle \\ \cdots \\ \langle f, \phi_n\rangle \end{bmatrix} \tag{9}$$

2.5 Escolha do sistema de coordenadas

Para este exercício defini-se o subespaço U_n por um espaço de Splines Lineares $S^0_{2,n}[0,1]$ com nós uniformemente espaçados em [0,1]. Tomando $h = 1/(n+1)$ e $x_i = ih, i = 0, 1, \ldots, n+1$ teremos:

$$S^0_{2,n}[0,1] = \left\{ s(x) \in C[0,1] : s(0) = s(1) = 0 \text{ e } \left. s \right|_{[x_i, x_{i+1}]} \in P_1 \right\} \tag{10}$$

Cada Spline em $S^0_{2,n}[0,1]$ é uma função contínua em [0,1] se anulando nos extremos e coincidindo com uma reta entre cada dois nós. De forma que cada Spline fica unicamente determinado através de seus nós $x_1 ... x_n$.

Formar uma base neste dito espaço vetorial requer funções "chápeus"$\phi_i(x)$ que valem 0 fora de $[x_{i-1}, x_{i+1}]$, $\phi_i(x) = (x - x_{i-1})/h$ em $[x_{i-1}, x_i]$ e $\phi_i(x) = (x_{i+1} - x)/h$ em $[x_i, x_{i+1}]$. O primeiro intervalo é chamado suporte da função ϕ_i pois fora dele a função se anula.Em elementos finitos, procura-se bases com suportes pequenos. Em resumo pode-se definir as funções ϕ:

$$\phi_i(x) = \begin{cases} 0, & 0 \leq x \leq x_{i-1} \\ \frac{x - x_{i-1}}{h_{i-1}}, & x_{i-1} < x \leq x_i \\ \frac{x_{i+1} - x}{h_i}, & x_i < x \leq x_{i+1} \\ 0, & x_{i+1} < x \leq 1 \end{cases} \tag{11}$$

A intersecção entre interiores de suportes ϕ_i e ϕ_j será não nula apenas se $|i - j| \leq 1$. Isso resulta que $\langle \phi_i, \phi_j\rangle_L = 0$ se $|i - j| > 1$. Isso torna o sistema (9) uma matriz tridiagonal.

Além disto tem-se que :

$$\langle f, \phi_i\rangle = \int_0^1 f(x)\phi_i(x)dx = \int_{x_{i-1}}^{x_{i+1}} f(x)\phi_i(x)dx \tag{12}$$

Observe ainda que $\phi_i'(x)$ é nula fora de $[x_{i-1}, x_{i+1}]$, vale $1/h$ em (x_{i-1}, x_i) e $-1/h$ em (x_i, x_{i+1}). No caso em que $k(x) = 1$ e $q(x) = 0$ a matriz do sistema é tridiagonal com valores $2/h$ na diagonal principal e $-1/h$ nas diagonais vizinhas a esta.

2.6 Montagem da Matriz e Solução do Sistema

Para efetuar a montagem da matriz, basta a avaliação dos produtos internos $\langle f, \phi_i \rangle$ e $\langle \phi_i, \phi_j \rangle_L$ - estes apenas para a diagonal principal e as diagonais secundárias inferior e superior.

Sendo os produtos internos $\langle \phi_i, \phi_i \rangle_L$ na diagonal principal dados por:

$$\begin{aligned} b_i &= \int_0^1 \left\{ p(x) \left[\phi_i'(x)\right]^2 + q(x) \left[\phi_i(x)\right]^2 \right\} dx \\ &= \left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} p(x)dx + \left(\frac{-1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} p(x)dx \\ &\quad + \left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} (x - x_{i-1})^2 q(x)dx + \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)^2 q(x)dx, \end{aligned} \tag{13}$$

Enquanto os produtos da diagonal superior $\langle \phi_i, \phi_{i+1} \rangle_L$(ou seja para i= 1,...,n-1):

$$\begin{aligned} c_i &= \int_0^1 \left\{ p(x)\phi_i'(x)\phi_{i+1}'(x) + q(x)\phi_i(x)\phi_{i+1}(x) \right\} dx \\ &= -\left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} p(x)dx + \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)(x - x_i) q(x)dx \end{aligned} \tag{14}$$

E, por fim os produtos internos da diagonal inferior $\langle \phi_i, \phi_{i-1} \rangle_L$ (ou seja, para i=1,...,n):

$$\begin{aligned} a_i &= \int_0^1 \left\{ p(x)\phi_i'(x)\phi_{i-1}'(x) + q(x)\phi_i(x)\phi_{i-1}(x) \right\} dx \\ &= -\left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} p(x)dx + \left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} (x_i - x)(x - x_{i-1}) q(x)dx \end{aligned} \tag{15}$$

E, por fim pode-se definir o produto entre f(x) e ϕ como:

$$d_i = \int_0^1 f(x)\phi_i(x)dx = \frac{1}{h_{i-1}} \int_{x_{i-1}}^{x_i} (x - x_{i-1}) f(x)dx + \frac{1}{h_i} \int_{x_i}^{x_{i+1}} (x_{i+1} - x) f(x)dx \tag{16}$$

Montado o sistema tridiagonal, resolve-se o sistema pelo método LU, obtendo uma solução $\bar{u}_n(x) = \sum_{i=1}^n \alpha_i \phi_i(x)$ de (8) $\forall v \in U_0$ que melhor aproxima a solução de $u(x)$ em (8) e consequentemente em (5). Ao aumentar-se o valor de n e reduzir o valor de h, refina-se a aproximação e, se u(x) for suficientemente diferenciável teremos que $\|\bar{u}_n(x) - u(x)\| = O(h^2)$.

2.7 Condições de Fronteiras Não Homogêneas e Intervalo

Para tratar o problema de fronteiras não homogêneas, trata-se o caso como um homogêneo pela seguinte equação:

$$L(v(x)) = f(x) + (b-a)k'(x) - q(x)(a + (b-a)x) = \tilde{f}(x), v(0) = v(1) = 0 \tag{17}$$

Neste caso pode-se mostrar que $u(x) = v(x) + a + (b-a)x$ é a solução da equação (5) com as condições de fronteiras não homogêneas $u(0) = a$ e $u(1) = b$.

$$L(v(x)) = f(x) + (b-a)k'(x) - q(x)(a + (b-a)x) = \tilde{f}(x) \tag{18}$$

para $v(0) = v(1) = 0$ tem-se

$$L(v(x)) = L(u(x)) + L(+a) + L((b-a)x) \tag{19}$$

Considerando $L(u(x)) = f(x), L(+a) = 0, L((b-a)x) = k'(x)(b-a) + q(x)(b-a)x$ e $u(x) = v(x) + a + (b-a)x$ tal que segue o seguinte resultado :

$$L(u(x)) = L(v(x)) + L(a) + L((b-a)x) \tag{20}$$

Provando que $L(u(x))$ é uma transformação linear e que a equação mencionada para $v(x)$ é solução para a equação de métodos finitos utilizada.

Quando o intervalo para a equação diferencial for [0, L], usamos splines lineares neste intervalo com nós igualmente espaçados tomando-se h = L/(n + 1) e xi = ih, i = 0, 1, ..., n + 1.

Para obter as expressões para a base formada por funções chapéu adaptadas para as condições de contorno não homogêneas , basta adaptar o valor de **h** . Assim ,substituindo x em [0,1] e L(u(x)) por x em [0,L] e $L(u(\frac{u}{L}))$ respectivamente teremos a transformação desejada.

Para $x \in (0,1)$; $u(0) = u(1) = 0$

$$L(u(x)) = (-k(x) \cdot u'(x))' + q(x) \cdot u(x) = f(x) \tag{21}$$

e para $x \in (0, L)$:

$$\begin{aligned} g(x) = L(u(\frac{x}{L})) &= (-k(\frac{x}{L}) \cdot u'(\frac{x}{L}))' \\ g(x) = -k'(\frac{x}{L}) \cdot \frac{1}{L} \cdot u'(\frac{x}{L}) &- k(\frac{x}{L}) \cdot u"(\frac{x}{L}) \cdot \frac{1}{L} \\ g(x) = L(u(\frac{x}{L})) &= \frac{L(u(x)}{L} \end{aligned} \tag{22}$$

Assim obtendo as relações necessárias para efetuar as bases das funções chapéu.

3 Implementação do Código

3.1 Algoritmo

O algoritmo desejado, descrito no enunciado do exercício programa, pede que se implemente um método para o cálculo da função de distribuição de calor para um chip de dimensões L x L x h em contato com uma parede isolante e um resfriador , utilizando o método de elementos finitos com o método de solução de Ritz-Raleigh para resolução do problema. Para tais tarefas faz-se uso dos dois exercícios programas efetuados anteriormente, para criação de uma matriz tridiagonal cujos termos são todos produtos internos equacionados utilizando o método de integração gaussiana e posteriormente, resolvendo a matriz com o método LU. Também elabora-se formas de trabalhar estas resoluções em condições não homogêneas e em diferentes intervalos.

De forma geral, é demandado no exercício além da resolução de duas tarefas de validação , as quais já contém a maioria das variáveis a ser usadas em sua forma mais simples, a fim de averiguar a funcionalidade do código. Em seguida, são propostas duas tarefas com vários testes possíveis em cada uma delas: a primeira para averiguar como diferenças na função Q_+ e Q_- podem afetar o padrão de distribuição geral de calor pelo chip; a segunda refere-se a situação de parâmetro de entrada não homogêneo - cria-se uma variação no K e consequentemente em ρ e C avaliando mais uma vez como estas diferenças causa interferências na distribuição de calor.

3.2 Metodologia aplicada

O algorítimo aplicado para a resolução desse exercício programa segue a seguinte linha de raciocínio:

- Em primeiro cria-se funções capazes de efetuar os produtos internos necessários para execução do método dos elementos finitos com o auxílio do código do exercício programa II, pela integração gaussiana.
- Em seguida, formam-se vetores com os dados das matrizes para resolução do sistema linear para se encontrar $\bar{u_n}(x)$ (solução mais próxima para a equação) com o auxílio do código do exercício programa I, método LU de resolução de matrizes.
- Por fim efetuam-se as partes para as particularidades mencionadas nas partes de equilíbrios tanto de forças quanto de material não homogêneo para diversas condições a fim de possibilitar as análises descritas na secção 4.

A seguir será destrinchado cada um destes seguimentos de raciocínio, subdividido pelas tarefas a serem realizadas: Validação, Equilíbrio com forçantes de Calor e Equilíbrio com variação de material.

3.3 Método de elementos finitos

Nesta primeira parte do código, dedicou-se a construir o método de resolução por elementos finitos, com base nos códigos antigos, mencionados na metodologia.

```
1 import numpy as np
2
3 # Resolve sistema tridiagonais
```

```
4  def sistemaTridiagLU(a, b, c, d, n):
5      """
6      a: vetor de coeficientes da diagonal secundaria inferior
7      b: vetor de coeficientes da diagonal principal
8      c: vetor de coeficientes da diagonal secundaria superior
9      d: vetor resultado
10     n: dimensão do sistema nxn
11
12     """
13     # Calcula vetores u e l
14     u = [b[0]]
15     l = []
16     for i in range(1, n):
17         l.append(a[i] / u[i - 1])
18         u.append(b[i] - l[i - 1] * c[i - 1])
19
20     # Calcula solução de L*y = d
21     y = [d[0]]
22     for i in range(1, n):
23         y.append(d[i] - l[i - 1] * y[i - 1])
24
25     # Calcula solução de U*x = y
26     x = [0] * n
27     x[n - 1] = y[n - 1] / u[n - 1]
28     for i in reversed(range(0, n - 1)):
29         x[i] = (y[i] - c[i] * x[i + 1]) / u[i]
30
31     return x
32
33
34 # Calculo da integral simples ou dupla
35 def integral_simples_ou_dupla(
36     f, a, b, pontos_e_pesos_x, c=lambda x: 0, d=lambda x: 1,
       pontos_e_pesos_y=None
37 ):
38     """
39     f: função de x e y a ser integrada
40     a: limite inferior do intervalo de integração em x
41     b: limite superior do intervalo de integração em x
42     pontos_e_pesos_x: pontos e pesos para o método de Gauss em x (
       deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
43     c: limite inferior do intervalo de integração em y (função de x)
44     d: limite superior do intervalo de integração em y (função de x)
45     pontos_e_pesos_y: pontos e pesos para o método de Gauss em y (
       deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
46
47     """
48
49     pontos_e_pesos_y = (
50         pontos_e_pesos_x if pontos_e_pesos_y is None else
       pontos_e_pesos_y
```

```
    )

    I = 0
    for x_i, w_i in pontos_e_pesos_x:
        # troca de variavel de x para o intervalo [a,b]
        x_i = (a + b) / 2 + (b - a) * x_i / 2
        F = 0
        for y_ij, w_ij in pontos_e_pesos_y:
            # troca de variavel de y para o intervalo [c(x_i),d(x_i)]
            y_ij = (c(x_i) + d(x_i)) / 2 + (d(x_i) - c(x_i)) * y_ij / 2
            F += w_ij * f(x_i, y_ij) * (d(x_i) - c(x_i)) / 2
        I += w_i * F * (b - a) / 2

    return I
```

O listing acima apresenta o início do código que consta basicamente na junção dos últimos exercícios programas para o método de solução de matrizes LU e o método de integração de Gauss para integrais duplas e simples.

Em seguida, trabalhou-se na primeira parte citada na metodologia: efetuar os produtos internos para montar o sistema (9), criando uma função para a resolução do produto interno da diagonal principal (13), outras duas para as diagonais secundárias superior (14) e inferior (14)e uma última para os produtos de f(x) e as funções $\phi(x)$ (16)- todos utilizando as funções do exercício programa dois para a integração de gauss para integral simples.

```
def calcula_produto_interno_phis_diagonal_principal(
    k, q, past_xi, current_xi, next_xi, h, pontos_e_pesos_phi
):
    """
    k: função k(x)
    q: função q(x)
    past_xi: xi[i-1], xi anterior
    current_xi: xi[i], xi atual
    next_xi: xi[i+1], xi posterior
    h: tamanho do intervalo
    pontos_e_pesos_phi: pontos e pesos para o método de Gauss em phi (deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)

    """

    # Calcula o produto interno de phii e phii
    phii_phii_1 = lambda x, y: k(x, y) + (x - past_xi) ** 2 * q(x, y)
    phii_phii_2 = lambda x, y: k(x, y) + (next_xi - x) ** 2 * q(x, y)
    return (1 / h) ** 2 * (
        integral_simples_ou_dupla(
            phii_phii_1, past_xi, current_xi, pontos_e_pesos_phi
        )  # intervalo de integração [past_xi, current_xi]
        + integral_simples_ou_dupla(
```

```
            phii_phii_2, current_xi, next_xi, pontos_e_pesos_phi
        )  # intervalo de integração [current_xi, next_xi]
    )


def calcula_produto_interno_phis_diagonal_secundaria_inferior(
    k, q, past_xi, current_xi, h, pontos_e_pesos_phi
):
    """
    k: função k(x)
    q: função q(x)
    past_xi: xi[i-1], xi anterior
    current_xi: xi[i], xi atual
    h: tamanho do intervalo
    pontos_e_pesos_phi: pontos e pesos para o método de Gauss em phi
    (deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
    """

    # Calcula o produto interno de phii e phii+1
    phii_phij = lambda x, y: k(x, y) + (current_xi - x) * (x -
   past_xi) * q(x, y)
    return -((1 / h) ** 2) * integral_simples_ou_dupla(
        phii_phij, past_xi, current_xi, pontos_e_pesos_phi
    )  # intervalo de integração [past_xi, current_xi]


def calcula_produto_interno_phis_diagonal_secundaria_superior(
    k, q, current_xi, next_xi, h, pontos_e_pesos_phi
):
    """
    k: função k(x)
    q: função q(x)
    current_xi: xi[i], xi atual
    next_xi: xi[i+1], xi posterior
    h: tamanho do intervalo
    pontos_e_pesos_phi: pontos e pesos para o método de Gauss em phi
    (deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
    """

    # Calcula o produto interno de phii e phii-1
    phij_phii = lambda x, y: k(x, y) + (next_xi - x) * (x -
   current_xi) * q(x, y)
    return -((1 / h) ** 2) * integral_simples_ou_dupla(
        phij_phii, current_xi, next_xi, pontos_e_pesos_phi
    )  # intervalo de integração [current_xi, next_xi]


def calcula_produto_interno_f_phi(
    f, past_xi, current_xi, next_xi, h, pontos_e_pesos_phi
):
    """
```

```
    f: função f(x)
    past_xi: xi[i-1], xi anterior
    current_xi: xi[i], xi atual
    next_xi: xi[i+1], xi posterior
    h: tamanho do intervalo
    pontos_e_pesos_phi: pontos e pesos para o método de Gauss em phi
    (deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
    """

    # Calcula o produto interno de f(x) e phii
    f_phii1 = lambda x, y: (x - past_xi) * f(x, y)
    f_phii2 = lambda x, y: (next_xi - x) * f(x, y)
    return (
        1
        / h
        * (
            integral_simples_ou_dupla(f_phii1, past_xi, current_xi,
pontos_e_pesos_phi)
            + integral_simples_ou_dupla(
                f_phii2, current_xi, next_xi, pontos_e_pesos_phi
            )
        )
    )


```

Feito os produtos internos, seguiu-se para uma função que armazena estes em vetores e utiliza a função *def sistemaTridiagLU* para efetuar a solução do sistema (9):

```
def sol_sistema_linear_tridiagonal(f, k, q, n, pontos_e_pesos_phi, L
=1):
    """
    f: função f(x)
    k: função k(x)
    q: função q(x)
    n: número de pontos
    pontos_e_pesos_phi: pontos e pesos para o método de Gauss em phi
    (deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
    L: opcional, define interval [0,L]
    """

    h = L / (n + 1)
    xi = [i * h for i in range(n + 2)]

    # Vetores do sistema linear (tridigonal)
    b = []  # diagonal principal
    a = []  # diagonal secundária inferior
    c = []  # diagonal secundária superior
    d = []  # vetor de termos independentes
    for i in range(n):
        past_xi = xi[i]
        current_xi = xi[i + 1]
```

```
        next_xi = xi[i + 2]

        b.append(
            calcula_produto_interno_phis_diagonal_principal(
                k, q, past_xi, current_xi, next_xi, h,
    pontos_e_pesos_phi
            )
        )
        a.append(

    calcula_produto_interno_phis_diagonal_secundaria_inferior(
                k, q, past_xi, current_xi, h, pontos_e_pesos_phi
            )
        )
        c.append(

    calcula_produto_interno_phis_diagonal_secundaria_superior(
                k, q, current_xi, next_xi, h, pontos_e_pesos_phi
            )
        )
        d.append(
            calcula_produto_interno_f_phi(
                f, past_xi, current_xi, next_xi, h,
    pontos_e_pesos_phi
            )
        )

    # Resolve o sistema linear
    alphas = np.array(sistemaTridiagLU(a, b, c, d, n))

    return alphas


```

Para obter os valores de cada uma das funções ϕ_i utilizou-se as equações descritas em (25) da seguinte forma:

```

def phi(x, xi, i, h):
    """
    x: valor de x
    xi: vetor de pontos
    i: índice do ponto de xi
    h: tamanho do intervalo
    """

    if xi[i - 1] < x <= xi[i]:
        return (x - xi[i - 1]) / h
    elif xi[i] < x <= xi[i + 1]:
        return (xi[i + 1] - x) / h
    else:
        return 0
```

```
16
17
```

Com os valores de α_i obtidos como solução do sistema, obtêm-se os valores de $\bar{u}_n(x)$ pela expressão $\bar{u}_n(x) = \sum_{i=1}^{n} \alpha_i \phi_i(x)$ e, em seguida é possível calcular o maior erro $\|\bar{u}_n - u\| = \max_{i=1,\ldots,n} |\bar{u}_n(x_i) - u(x_i)|$.

```
1  def u_barra(x, alphas, xi, h):
2      """
3      x: valor de x
4      alphas: vetor de alphas
5      xi: vetor de pontos
6      h: tamanho do intervalo
7
8      """
9
10     u_barra = 0
11     for i in range(len(alphas)):
12         u_barra += alphas[i] * phi(x, xi, i + 1, h)
13     return u_barra
14
15
16
17 def maior_erro(n, u_exato, alphas, L=1):
18     """
19     n: número de pontos
20     u_exato: função u(x)
21     alphas: vetor de alphas
22     L: opcional, define interval [0,L]
23     """
24
25     h = L / (n + 1)
26     xi = [i * h for i in range(n + 2)]
27     erro_max = 0
28     u_barra_erro_max = 0
29     u_exato_erro_max = 0
30     for x in xi:
31         if abs(u_exato(x) - u_barra(x, alphas, xi, h)) > erro_max:
32             erro_max = abs(u_exato(x) - u_barra(x, alphas, xi, h))
33             u_barra_erro_max = u_barra(x, alphas, xi, h)
34             u_exato_erro_max = u_exato(x)
35     return erro_max, u_barra_erro_max, u_exato_erro_max
```

Para o caso não homogêneo $u(0) = a$ e $u(1) = b$ temos $\bar{u}_n^{a,b}(x) = \bar{u}_n(x) + \frac{a+(b-a)x}{L}$

```
1  def u_barra_nao_homogenea(x, a, b, alphas, xi, h, L=1):
2      """
3      x: valor de x
4      a: valor de u(0)
5      b: valor de u(L)
6      alphas: vetor de alphas
7      xi: vetor de pontos
8      h: tamanho do intervalo
```

```
9
10      """
11      return u_barra(x, alphas, xi, h) + (a + (b - a) * x) / L
```

3.4 Modo I

Determinadas todas as funções necessárias para a implementação do método de elementos finitos, separou-se o código principal em 4 modos. O primeiro deles é dedicado a realizar a seguinte validação do método proposto:

4.2 Validação

No primeiro passo do projeto você deve implementar o método de elementos finitos (veja Seção 3) para resolver a equação (4) com $q(x) \equiv 0$, no intervalo $[0, L]$ e condições de contorno $u(0) = a$, $u(L) = b$. Teste o programa com o exemplo no intervalo $[0, 1]$ onde $k(x) = 1$, $q(x) = 0$, $f(x) = 12x(1 - x) - 2$, com condições de contorno homogênias. Neste caso, a solução exata é $u(x) = x^2(1 - x)^2$. Verifique que a convergência do método é de segunda ordem[2] calculando as aproximações com $n = 7$, 15, 31 e 63, avaliando em cada caso $\|\bar{u}_n - u\| = \max_{i=1,\ldots,n} |\bar{u}_n(x_i) - u(x_i)|$. Descreva esta análise no relatório.

Figura 1: Enunciado primeiro exercício de validação

```
1  if modo == 1:
2      print(
3          "\nModo 1 - Validação no intervalo [0,1] com k(x)=1, q(x)=0,
       f(x)=12x(1-x)-2"
4      )
5      print("\nSolução exata u(x) = (x**2 *(1 - x)**2")
6      print("\nParametros: \nk(x)=1, \nq(x)=0, \nf(x)=12x(1-x)-2")
7
8      input("\n\nAvaliando o máximo erro nos pontos xi para diferentes
       n's:")
9
10     for n in [7, 15, 31, 63]:  # números de pontos
11
12         print(f"\nn={n} -->")
13
14         f = lambda x, y: 12 * x * (1 - x) - 2  # função f(x)
15         k = lambda x, y: 1  # função k(x)
16         q = lambda x, y: 0  # função q(x)
17         u_exato = lambda x: x**2 * (1 - x) ** 2  # solução exata
18         alphas = sol_sistema_linear_tridiagonal(
19             f, k, q, n, n10
20         )  # solução do sistema
21
22         erro_max, u_barra_erro_max, u_exato_erro_max = maior_erro(
23             n, u_exato, alphas
24         )
25         print(
26             f"u_barra = {u_barra_erro_max:.22f} ; u_exato = {
       u_exato_erro_max:.22f}; erro = {erro_max:.22f}"
27         )
```

```
28
29      input("\nPressione Enter para ver todos os erros calculados.")
30      for n in [7, 15, 31, 63]:
31
32          print(f"\nn={n} —>")
33
34          alphas = sol_sistema_linear_tridiagonal(
35              f, k, q, n, n10
36          )  # solução do sistema
37          L = 1
38          h = L / (n + 1)
39          xi = [i * h for i in range(n + 2)]
40          u_barra_lista = []
41          for x in xi:
42              u_barra_lista.append(u_barra(x, alphas, xi, h))
43              print(
44                  f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f} ; u_exato = {u_exato(x):.22f}; erro = {u_barra(x,alphas,xi
    ,h)-u_exato(x):.22f}"
45              )
46          u_exato_lista = [u_exato(x) for x in np.linspace(0, 1)]
47
48          plt.plot(xi, u_barra_lista, label="$u$")
49          plt.plot(np.linspace(0, 1), u_exato_lista, label="$\overline
    {u}_n$")
50          plt.legend()
51          plt.grid()
52          plt.ylabel("$u(x)$")
53          plt.xlabel("$x$")
54          plt.show()
55
56      input("\n\nAvaliando o erro nos pontos p=(x[i-1]-xi)/2 para
    diferentes n's:")
57
58      for n in [7, 15, 31, 63]:  # números de pontos
59
60          print(f"\nn={n} —>")
61
62          f = lambda x, y: 12 * x * (1 - x) - 2  # função f(x)
63          k = lambda x, y: 1  # função k(x)
64          q = lambda x, y: 0  # função q(x)
65          u_exato = lambda x: x**2 * (1 - x) ** 2  # solução exata
66          alphas = sol_sistema_linear_tridiagonal(
67              f, k, q, n, n10
68          )  # solução do sistema
69
70          h = L / (n + 1)
71          xi = [i * h for i in range(n + 2)]
72
73          # testanto em nos pontos pi
74          P_i = [0.33, 0.47, 0.66, 0.72]
```

```
75          for x in P_i:
76              print(
77                  f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f} ; u_exato = {u_exato(x):.22f}; erro = {u_barra(x,alphas,xi
    ,h)-u_exato(x):.22f}"
78              )
```

O modo é dividido em 3 partes. Primeiramente, são calculados os erros máximos para cada n. Depois, todos o cálculo dos erros de cada n para cada ponto x_i é impresso, juntamento com um gráfico comparando a solução obtida com a solução exata. Por fim, são calculados os valores e erros para os pontos $P_i = [0.33, 0.47, 0.66, 0.72]$

3.5 Modo II

Para o segundo modo, trabalhou-se em realizar uma segunda validação, sugeria à parte do enunciado do exercício:

Complemento para a Seção 4.2

Para verificar se o seu código está funcionando corretamente, sugerimos o seguinte teste adicional para a resolução da equação (4):

Teste o programa com o exemplo no intervalo $[0, 1]$ onde $k(x) = e^x$, $q(x) = 0$, $f(x) = e^x + 1$, com condições de contorno homogênias. Neste caso, a solução exata é $u(x) = (x-1)(e^{-x}-1)$. Verifique que a convergência do método é de segunda ordem[1] calculando as aproximações com $n = 7$, 15, 31 e 63, avaliando em cada caso $\|\bar{u}_n - u\| = \max_{i=1,\ldots,n} |\bar{u}_n(x_i) - u(x_i)|$. Descreva esta análise no relatório.

Figura 2: Enunciado exercício extra de validação

```
1  elif modo == 2:
2      print(
3          "\nModo 2 - Validação no intervalo [0,1] com k(x)=e**x, q(x)
       =0, f(x)=e**(x)+1"
4      )
5      print("\nSolução exata u(x) = (x-1)*(e**(-x) - 1)")
6      print("\nParametros: \nk(x)=1, \nq(x)=0, \nf(x)=e**(x)+1")
7
8      input("\n\nAvaliando o máximo erro para diferentes n's:")
9
10     for n in [7, 15, 31, 63]:  # números de pontos
11
12         print(f"\nn={n} -->")
13
14         f = lambda x, y: np.exp(x) + 1  # função f(x)
15         k = k = lambda x, y: np.exp(x)  # função k(x)
16         q = lambda x, y: 0  # função q(x)
17         u_exato = lambda x: (x - 1) * (np.exp(-x) - 1)  # solução
       exata
18         alphas = sol_sistema_linear_tridiagonal(
19             f, k, q, n, n10
20         )  # solução do sistema
21
```

```
        erro_max, u_barra_erro_max, u_exato_erro_max = maior_erro(
            n, u_exato, alphas
        )
        print(
            f"u_barra = {u_barra_erro_max:.22f} ; u_exato = {
    u_exato_erro_max:.22f}; erro = {erro_max:.22f}"
        )

    input("\nPressione Enter para ver todos os erros calculados.")
    for n in [7, 15, 31, 63]:

        print(f"\nn={n} -->")

        alphas = sol_sistema_linear_tridiagonal(
            f, k, q, n, n10
        )  # solução do sistema
        L = 1
        h = L / (n + 1)
        xi = [i * h for i in range(n + 2)]
        u_barra_lista = []
        for x in xi:
            u_barra_lista.append(u_barra(x, alphas, xi, h))
            print(
                f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f} ; u_exato = {u_exato(x):.22f}; erro = {u_barra(x,alphas,xi
    ,h)-u_exato(x):.22f}"
            )
        u_exato_lista = [u_exato(x) for x in np.linspace(0, 1)]

        plt.plot(xi, u_barra_lista, label="$u$")
        plt.plot(np.linspace(0, 1), u_exato_lista, label="$\overline
    {u}_n$")
        plt.legend()
        plt.grid()
        plt.ylabel("$u(x)$")
        plt.xlabel("$x$")
        plt.show()
```

O modo é dividido em 2 partes. Primeiramente, são calculados os erros máximos para cada n. Depois, todos o cálculo dos erros de cada n para cada ponto x_i é impresso juntamento com um gráfico comparando a solução obtida com a solução exata.

3.6 Modo III

Nesta parte , buscou-se utilizar as funções criadas e validadas para resolução da seguinte situação apontada

4.3 Equilíbrio com forçantes de calor

Vamos considerar que o chip seja formado apenas de silício ($k(x) = k = 3,6W/(mK)$), considerando que há produção de calor pelo chip e que exista resfriamento. Vamos assumir que o chip esquenta mais em sua parte central que nas bordas, o que pode ser modelado por uma Gaussiana da seguinte forma,

$$Q_+(x) = Q_+^0 e^{-(x-L/2)^2/\sigma^2} \tag{9}$$

com Q_+^0 uma constante indicando o máximo de calor gerado no centro do chip e σ controlando a variação de geração de calor em torno do ponto central do chip. Se σ for muito pequeno, podemos ter uma calor gerado praticamente somente no centro do chip.

Quanto ao resfriamento, podemos modelar de forma análoga, ou, por exemplo, assumir que o resfriamento se dá de forma uniforme ($Q_-(x) = Q_-^0$ constante), ou ainda que o resfriamento seja mais intenso próximo dos extremos, usando

$$Q_-(x) = Q_-^0 \left(e^{-(x)^2/\theta^2} + e^{-(x-L)^2/\theta^2} \right). \tag{10}$$

Figura 3: Enunciado equilíbrio de forçantes de calor

Como mencionado na seção 2 deste relatório, podemos estudar o comportamento da distribuição de calor no conjunto chip e resfriador separando funções de calor gerado pelo chip Q_+ e o calor retirado pelo resfriador Q_- , cuja soma , em equilíbrio deve sempre se igualar a zero.

O exercício nos propõem a seguinte função de calor:

$$Q(x) = Q_+(x) - Q_-(x) \tag{23}$$

Onde

$$\begin{aligned} Q_+(x) &= Q_+^0 e^{-(x-L/2)^2/\sigma^2} \\ Q_-(x) &= Q_-^0 \left(e^{-(x)^2/\theta^2} + e^{-(x-L)^2/\theta^2} \right) \end{aligned} \tag{24}$$

De tal forma que é possível tecer análises sobre o comportamento da distribuição e retirada de calor no chip variando os parâmetros $Q_+^0, Q_-^0, \sigma, \theta$. Os casos estudados estão documentados em gráficos dispostos na Seção 4 deste relatório e incluem , para condições de contorno homogêneas, como se comporta funções de calor constantes (variando apenas o valor de Q_+^0, Q_-^0), e não constantes (avaliando o efeito de σ e θ na distribuição de calor).

Portanto, este modo consiste em várias pequenas resoluções, variando levemente os parâmetros de entrada. Os primeiros quatro casos são os em que as funções Q são constantes - $Q_+(x) = Q_+^0$ e $Q_-(x) = Q_-^0$ - variando-se os valores para os casos

$$Q(x) = \begin{cases} Q_+(x) - Q_-(x) = 2000, & \text{ou seja } Q_+(x) \text{ maior que } Q_-(x) \\ Q_+(x) - Q_-(x) = -2000, & \text{ou seja } Q_-(x) \text{ maior que } Q_+(x) \\ Q_+(x) - Q_-(x) = 20000, & \text{ou seja } Q_+(x) \text{ muito maior que } Q_-(x) \\ Q_+(x) - Q_-(x) = -20000, & \text{ou seja } Q_-(x) \text{ muito maior que } Q_+(x) \end{cases} \tag{25}$$

```
1  elif modo == 3:
2      print("\nModo 3 - Equilíbrio com forçantes de calor")
3      print("\nParametros: \nk(x)=3.6, \nq(x)=0, \nf(x)=Q(x)")
4      print(
5          "\nSendo Q(x) representa a soma do calor gerado pelo chip (Q
       +) e o calor retirado pelo resfriador (Q-)"
6      )
```

```
    print("\n\nVariando parametros:")

    print(
        "\n  Q+ e Q- constantes e Q+ > Q- (portanto Q(x) constante e maior que zero)"
    )
    input(" Parametros: Q+ - Q- = 2000")

    n = 63
    f = lambda x, y: 2000  # função f(x)
    k = lambda x, y: 3.6  # função k(x)
    q = lambda x, y: 0  # função q(x)
    alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solução do sistema

    L = 1
    h = L / (n + 1)
    xi = [i * h for i in range(n + 2)]
    u_barra_lista = []
    for x in xi:
        u_barra_lista.append(u_barra(x, alphas, xi, h))
        print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f}")

    plt.plot(xi, u_barra_lista, label="$T(x)$")
    plt.legend()
    plt.grid()
    plt.show()

    print("\n\n  Q+ e Q- constantes e Q+ >>> Q-")
    input(" Parametros: Q+ - Q- = 20000")

    n = 63
    f = lambda x, y: 20000  # função f(x)
    k = lambda x, y: 3.6  # função k(x)
    q = lambda x, y: 0  # função q(x)
    alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solução do sistema

    L = 1
    h = L / (n + 1)
    xi = [i * h for i in range(n + 2)]
    u_barra_lista = []
    for x in xi:
        u_barra_lista.append(u_barra(x, alphas, xi, h))
        print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f}")

    plt.plot(xi, u_barra_lista, label="$T(x)$")
    plt.legend()
    plt.grid()
```

```
53      plt.show()
54
55      print(
56          "\n\n  Q+ e Q- constantes e Q+ < Q- (portanto Q(x) constante
        e menor que zero)"
57      )
58      input(" Parametros: Q+ - Q- = -2000")
59
60      n = 63
61      f = lambda x, y: -2000  # função f(x)
62      k = lambda x, y: 3.6  # função k(x)
63      q = lambda x, y: 0  # função q(x)
64      alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solu
        ção do sistema
65
66      L = 1
67      h = L / (n + 1)
68      xi = [i * h for i in range(n + 2)]
69      u_barra_lista = []
70      for x in xi:
71          u_barra_lista.append(u_barra(x, alphas, xi, h))
72          print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f
        }")
73
74      plt.plot(xi, u_barra_lista, label="$T(x)$")
75      plt.legend()
76      plt.grid()
77      plt.show()
78
79      print("\n\n  Q+ e Q- constantes e Q+ <<< Q-")
80      input(" Parametros: Q+ - Q- = -20000")
81
82      n = 63
83      f = lambda x, y: -20000  # função f(x)
84      k = lambda x, y: 3.6  # função k(x)
85      q = lambda x, y: 0  # função q(x)
86      alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solu
        ção do sistema
87
88      L = 1
89      h = L / (n + 1)
90      xi = [i * h for i in range(n + 2)]
91      u_barra_lista = []
92      for x in xi:
93          u_barra_lista.append(u_barra(x, alphas, xi, h))
94          print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f
        }")
95
96      plt.plot(xi, u_barra_lista, label="$T(x)$")
97      plt.legend()
98      plt.grid()
```

```
99      plt.show()
```

Então, segue-se com resoluções onde apenas a expressão de $Q_+(x)$ é considerada, com:

$$Q(x) = Q_+(x) = Q_+^0 e^{-(x-L/2)^2/\sigma^2} \tag{26}$$

Sendo $Q_+^0 = 2000$, $L = 1$, variando os valores de $\sigma = 0.05, 0.1, 0.25, 0.5, 1, 5$.

```
print("\n\n  Q+ modelado por Q+(x) = Q+0 * e**-(x-L/2)**2/sigma**2 e
    Q-=0.")
print("  Variando sigma")
input("\n  Parametros: Q+0 = 2000, L=1, sigma = variando, Q-0 = 0")
n = 63
Q0 = 2000
sigmas = [0.05, 0.1, 0.25, 0.5, 1, 5]
L = 1
Q_0 = 0
for sigma in sigmas:
    f = (
        lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma**2) -
    Q_0
    )  # função f(x)
    k = lambda x, y: 3.6  # função k(x)
    q = lambda x, y: 0  # função q(x)
    alphas = sol_sistema_linear_tridiagonal(
        f, k, q, n, n10
    )  # solução do sistema

    h = L / (n + 1)
    xi = [i * h for i in range(n + 2)]
    u_barra_lista = []
    for x in xi:
        u_barra_lista.append(u_barra(x, alphas, xi, h))
        print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f
   }")

    plt.plot(xi, u_barra_lista, label="$\sigma=$" + str(sigma))
plt.legend()
plt.grid()
plt.show()
```

Segue-se para resoluções onde apenas a expressão de $Q_-(x)$ é considerada, com:

$$Q(x) = Q_-(x) = Q_-^0 \left(e^{-(x)^2/\theta^2} + e^{-(x-L)^2/\theta^2}\right) \tag{27}$$

Sendo $Q_-^0 = 2000$, $L = 1$, variando os valores de $\theta = 0.05, 0.1, 0.25, 0.5, 1, 5$.

```
print(
    "\n\n  Q+=0 e Q- modelado por Q-(x) = Q-0 * (e**(-x**2/  **2) +
   e**(-(x-L)**2/  **2))"
)
print("Variando theta")
input("\n  Parametros: Q+0 = 0, L=1, theta = variando, Q-0 = 2000")
```

```
6
7  n = 63
8  Q0 = 0
9  thetas = [0.05, 0.1, 0.25, 0.5, 1, 5]
10 L = 1
11 Q_0 = 2000
12 for theta in thetas:
13     f = lambda x, y: Q0 - (
14         Q_0
15         * (
16             np.e ** (-(x**2) / theta**2)
17             + np.e ** (-((x - L) ** 2) / theta**2)
18         )
19     )  # função f(x)
20     k = lambda x, y: 3.6  # função k(x)
21     q = lambda x, y: 0  # função q(x)
22     alphas = sol_sistema_linear_tridiagonal(
23         f, k, q, n, n10
24     )  # solução do sistema
25
26     L = 1
27     h = L / (n + 1)
28     xi = [i * h for i in range(n + 2)]
29     u_barra_lista = []
30     for x in xi:
31         u_barra_lista.append(u_barra(x, alphas, xi, h))
32         print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f
       }")
33
34     plt.plot(xi, u_barra_lista, label="$\u03B8=$" + str(theta))
35 plt.legend()
36 plt.grid()
37 plt.show()
```

Segue-se para resoluções onde ambas as expressões de $Q_+(x)$ e $Q_-(x)$ é considerada, com:

$$Q(x) = Q_+(x) - Q_-(x)$$

$$Q(x) = Q_+^0 e^{-(x-L/2)^2/\sigma^2} - Q_-^0 \left(e^{-(x)^2/\theta^2} + e^{-(x-L)^2/\theta^2}\right)$$

Sendo $Q_+^0 = 2000$, $Q_-^0 = 1000$, $L = 1$, variando os valores de $\theta = 0.1, 5$ e de $\sigma = 0.1, 5$.

```
1 print(
2     "\n\n  Q+ modelado por Q+(x) = Q+0 * e**-(x-L/2)**2/sigma**2 e Q
      - modelado por Q-(x) = Q-0 * (e**(-x**2/  **2) + e**(-(x-L)**2/
      /**2))."
3 )
4 print("Variando sigma e theta.")
5 input(
6     "\n  Parametros: Q+0 = 2000, L=1, sigma = variando, theta =
      variando, Q-0 = 1000"
```

```
)

n = 63
L = 1
Q0 = 2000
thetas = [0.1, 5]
sigmas = [0.1, 5]
L = 1
Q_0 = 1000
for theta in thetas:
    for sigma in sigmas:
        f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma **2) - (
            Q_0
            * (
                np.e ** (-(x**2) / theta**2)
                + np.e ** (-((x - L) ** 2) / theta**2)
            )
        )  # função f(x)
        k = lambda x, y: 3.6  # função k(x)
        q = lambda x, y: 0  # função q(x)
        alphas = sol_sistema_linear_tridiagonal(
            f, k, q, n, n10
        )  # solução do sistema

        h = L / (n + 1)
        xi = [i * h for i in range(n + 2)]
        u_barra_lista = []
        for x in xi:
            u_barra_lista.append(u_barra(x, alphas, xi, h))
            print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f}")

        plt.plot(
            xi,
            u_barra_lista,
            label="$\u03B8=$" + str(theta) + " $\sigma=$" + str(
    sigma),
        )
plt.legend()
plt.grid()
plt.show()
```

Por fim, a última sequência de análises é feita comparando o comportamento da distribuição de calor para situações não homogêneas (intervalo de [0,L] e u(x) não nulo nos contornos), com $Q(x)$ dado pelas expressões de Q_+ e Q_-

Considerou-se, $Q_+^0 = 2000$, $Q_-^0 = 1000$, $\theta = 0.1$, $\sigma = 0.1$, $T(0) = 298.15$, variando $T(L) = 298.15$, 328.15, 398.15 e $L = 0.1$, 0.5, 1, 5.

```
print(
    "\n\n  Q+ modelado por Q+(x) = Q+0 * e**-(x-L/2)**2/sigma**2 e Q
    - modelado por Q-(x) = Q-0 * (e**(-x**2/  **2) + e**(-(x-L)**2/
```

```
    /**2)).")
)
print("Variando condições de contorno e L")
input(
    "\n  Parametros: Q+0 = 2000, L=1, sigma = 0.1, theta = 0.1, Q-0 = 1000, T(0)=298.15, T(L) = 389.15"
)

T0 = 298.15
T1s = [298.15, 328.15, 398.15]
n = 63
Ls = [0.1, 0.5, 1, 5]
Q0 = 2000
theta = 0.1
sigma = 0.1
Q_0 = 1000
for L in Ls:
    f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma**2) - (
        Q_0
        * (
            np.e ** (-(x**2) / theta**2)
            + np.e ** (-((x - L) ** 2) / theta**2)
        )
    )  # função f(x)
    k = lambda x, y: 3.6  # função k(x)
    q = lambda x, y: 0  # função q(x)
    alphas = sol_sistema_linear_tridiagonal(
        f, k, q, n, n10
    )  # solução do sistema

    h = L / (n + 1)
    xi = [i * h for i in range(n + 2)]
    for T1 in T1s:
        u_barra_lista = []
        for x in xi:
            u_barra_lista.append(
                u_barra_nao_homogenea(x, T0, T1, alphas, xi, h)
            )
            print(
                f"x = {x:.4f} ; u_barra = {u_barra_nao_homogenea(x, T0,T1,alphas,xi,h):.22f}"
            )

        plt.plot(
            xi,
            u_barra_lista,
            label="$T(0)=$" + str(T0) + " e $T(L)=$" + str(T1),
        )
    plt.legend()
    plt.grid()
```

```
50      plt.show()
```

Todas essas soluções possibilitaram a criação dos gráficos e análises elaboradas na conclusão deste relatório.

3.7 Modo IV

Nesta parte, mais uma vez se utiliza dos métodos elaborados neste exercício programa para resoluções na seguinte situação-problema:

4.4 Equilíbrio com variação de material

Suponha agora que no bloco do processador tenhamos o chip, formado de silício, envolto por outro material. Isso faz com que k dependa de x, por exemplo como

$$k(x) = \begin{cases} k_s, \text{se } x \in (L/2 - d, L/2 + d), \\ k_a, \text{caso contrário,} \end{cases} \tag{11}$$

sendo k_s a condutividade térmica do silício e k_a a do material que envolve o chip e forma o bloco.

Usando o seu código de elementos finitos você pode verificar, por exemplo, o que acontece se o material que envolve o chip for alumínio ($k_a = 60W/mK$), ou outros materiais. Inclua essa análise no relatório.

Figura 4: Enunciado equilíbrio com variação de material

Para conseguir elaborar uma análise sobre como a variação do material afeta o resultado na distribuição de calor escolheram-se outros 2 materiais para serem testados além do alumínio sugerido um com valores de k muito inferior ao silício (espuma de poliestireno $K_e = 0,03W/(mK)$) e outra muito superior (prata $K_p = 429W/(mK)$).

Além disso variou-se também o tamanho que está faixa de outro material, variando d para um valor relativamente fino (um décimo do comprimento) e grosso (um terço do comprimento).

O código implementado segue a seguinte ordem: primeiro efetuar o cálculo para k externo com d= L/3 e depois com d= L/10 na ordem alumínio, prata e espuma de poliestireno. Para todas as contas utilizou-se a função $Q(x) = Q_+(x) - Q_-(x)$ com ambos σ e θ igual a 1, $Q_+(x) = 2000$, $Q_-(x) = 1000$, $L = 1$ e n=63. Todos os resultados são impressos junto com seus respectivos gráficos.

```
1  elif modo == 4:
2      print("\nModo 4 - Equilibrio com variacao de material")
3      print("\nAnalisando caso em que k(x)=k com k variando")
4      input("\nParametros: \n L=1, q(x)=0, theta=1, sigma=1, Q+0=2,Q
       -0=2 f(x)=Q(x)")
5
6      n = 63
7      ks = [0.5, 1, 3.6, 36]
8      Q0 = 2000
9      theta = 0.1
10     sigma = 0.1
11     L = 1
12     Q_0 = 1000
13     f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma**2) -
       (
14         Q_0
```

```
        * (
            np.e ** (-(x**2) / theta**2)
            + np.e ** (-((x - L) ** 2) / theta**2)
        )
    )  # função f(x)
    q = lambda x, y: 0  # função q(x)
    for k in ks:
        kx = lambda x, y: k  # função k(x)
        alphas = sol_sistema_linear_tridiagonal(
            f, kx, q, n, n10
        )  # solução do sistema

        L = 1
        h = L / (n + 1)
        xi = [i * h for i in range(n + 2)]
        u_barra_lista = []
        for x in xi:
            u_barra_lista.append(u_barra(x, alphas, xi, h))
            print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
:.22f}")

        plt.plot(xi, u_barra_lista, label="$k= $" + str(k))
    plt.grid()
    plt.legend()
    plt.ylabel("$T(x)$")
    plt.xlabel("$x$")
    plt.show()

    print("\nAnalisando para k(x)=ks se L/2-d<x<l/2+d e k(x)=ka caso
    contrario")
    input("\nParametros: \nks=3.6, ka=60, L=1, d=L/10, q(x)=0, \nf(x
)=Q(x)")

    n = 63
    Q0 = 2000
    theta = 0.1
    sigma = 0.1
    L = 1
    Q_0 = 1000
    f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma**2) -
    (
        Q_0
        * (
            np.e ** (-(x**2) / theta**2)
            + np.e ** (-((x - L) ** 2) / theta**2)
        )
    )  # função f(x)

    def k(x, y):  # função k(x)
        L = 1
        d = L / 10  # L/3
```

```
        ka = 60
        ks = 3.6
        if L / 2 - d < x < L / 2 + d:
            return ks
        else:
            return ka

    q = lambda x, y: 0  # função q(x)
    alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solução do sistema

    L = 1
    h = L / (n + 1)
    xi = [i * h for i in range(n + 2)]
    u_barra_lista = []
    for x in xi:
        u_barra_lista.append(u_barra(x, alphas, xi, h))
        print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f}")

    plt.plot(xi, u_barra_lista, label="$d=L/10$, $k_a=60$")

    print("\nParametros: \nks=3.6, ka=60, L=1, d=L/3, q(x)=0, \nf(x)=Q(x)")
    input("\nk(x)=ks se L/2-d<x<l/2+d e k(x)=ka caso contrario")

    n = 63
    Q0 = 2000
    theta = 0.1
    sigma = 0.1
    L = 1
    Q_0 = 1000
    f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma**2) - (
        Q_0
        * (
            np.e ** (-(x**2) / theta**2)
            + np.e ** (-((x - L) ** 2) / theta**2)
        )
    )  # função f(x)  # função f(x)

    def k(x, y):  # função k(x)
        L = 1
        d = L / 3  # L/3
        ka = 60
        ks = 3.6
        if L / 2 - d < x < L / 2 + d:
            return ks
        else:
            return ka

```

```
    q = lambda x, y: 0  # função q(x)
    alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solu
ção do sistema

    L = 1
    h = L / (n + 1)
    xi = [i * h for i in range(n + 2)]
    u_barra_lista = []
    for x in xi:
        u_barra_lista.append(u_barra(x, alphas, xi, h))
        print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f
}")

    plt.plot(xi, u_barra_lista, label="$d=L/3$, $k_a=60$")
    plt.legend()
    plt.grid()
    plt.ylabel("$T(x)$")
    plt.xlabel("$x$")
    plt.show()

    input("\nParametros: \nks=3.6, ka=429, L=1, d=L/10, q(x)=0, \nf(
x)=Q(x)")

    n = 63
    Q0 = 2000
    theta = 0.1
    sigma = 0.1
    L = 1
    Q_0 = 1000
    f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma**2) -
    (
        Q_0
        * (
            np.e ** (-(x**2) / theta**2)
            + np.e ** (-((x - L) ** 2) / theta**2)
        )
    )  # função f(x)  # função f(x)

    def k(x, y):  # função k(x)
        L = 1
        d = L / 10  # L/3
        ka = 429
        ks = 3.6
        if L / 2 - d < x < L / 2 + d:
            return ks
        else:
            return ka

    q = lambda x, y: 0  # função q(x)
    alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solu
ção do sistema
```

```
155
156     L = 1
157     h = L / (n + 1)
158     xi = [i * h for i in range(n + 2)]
159     u_barra_lista = []
160     for x in xi:
161         u_barra_lista.append(u_barra(x, alphas, xi, h))
162         print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f
    }")
163
164     plt.plot(xi, u_barra_lista, label="$d=L/10$, $k_a=429$")
165
166     print("\nParametros: \nks=3.6, ka=429, L=1, d=L/3, q(x)=0, \nf(x
    )=Q(x)")
167     input("\nk(x)=ks se L/2-d<x<l/2+d e k(x)=ka caso contrario")
168
169     n = 63
170     Q0 = 2000
171     theta = 0.1
172     sigma = 0.1
173     L = 1
174     Q_0 = 1000
175     f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma**2) -
        (
176         Q_0
177         * (
178             np.e ** (-(x**2) / theta**2)
179             + np.e ** (-((x - L) ** 2) / theta**2)
180         )
181     )  # função f(x)  # função f(x)
182
183     def k(x, y):  # função k(x)
184         L = 1
185         d = L / 3  # L/3
186         ka = 429
187         ks = 3.6
188         if L / 2 - d < x < L / 2 + d:
189             return ks
190         else:
191             return ka
192
193     q = lambda x, y: 0  # função q(x)
194     alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solu
    ção do sistema
195
196     L = 1
197     h = L / (n + 1)
198     xi = [i * h for i in range(n + 2)]
199     u_barra_lista = []
200     for x in xi:
201         u_barra_lista.append(u_barra(x, alphas, xi, h))
```

```
202         print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f
    }")
203
204     plt.plot(xi, u_barra_lista, label="$d=L/3$, $k_a=429$")
205     plt.legend()
206     plt.grid()
207     plt.ylabel("$T(x)$")
208     plt.xlabel("$x$")
209     plt.show()
210
211     input("\nParametros: \nks=3.6, ka=0.03, L=1, d=L/10, q(x)=0, \nf
    (x)=Q(x)")
212
213     n = 63
214     Q0 = 2000
215     theta = 0.1
216     sigma = 0.1
217     L = 1
218     Q_0 = 1000
219     f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma**2) -
        (
220         Q_0
221         * (
222             np.e ** (-(x**2) / theta**2)
223             + np.e ** (-((x - L) ** 2) / theta**2)
224         )
225     )  # função f(x)  # função f(x)
226
227     def k(x, y):  # função k(x)
228         L = 1
229         d = L / 10  # L/3
230         ka = 0.03
231         ks = 3.6
232         if L / 2 - d < x < L / 2 + d:
233             return ks
234         else:
235             return ka
236
237     q = lambda x, y: 0  # função q(x)
238     alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solu
    ção do sistema
239
240     L = 1
241     h = L / (n + 1)
242     xi = [i * h for i in range(n + 2)]
243     u_barra_lista = []
244     for x in xi:
245         u_barra_lista.append(u_barra(x, alphas, xi, h))
246         print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f
    }")
247
```

```
248     plt.plot(xi, u_barra_lista, label="$d=L/10$, $k_a=0.03$")
249
250     print("\nParametros: \nks=3.6, ka=0.03, L=1, d=L/3, q(x)=0, \nf(
        x)=Q(x)")
251     input("\nk(x)=ks se L/2-d<x<l/2+d e k(x)=ka caso contrario")
252
253     n = 63
254     Q0 = 2000
255     theta = 0.1
256     sigma = 0.1
257     L = 1
258     Q_0 = 1000
259     f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma**2) -
        (
260         Q_0
261         * (
262             np.e ** (-(x**2) / theta**2)
263             + np.e ** (-((x - L) ** 2) / theta**2)
264         )
265     )  # função f(x)  # função f(x)
266
267     def k(x, y):  # função k(x)
268         L = 1
269         d = L / 3  # L/3
270         ka = 0.03
271         ks = 3.6
272         if (L / 2 - d) < x and x < (L / 2 + d):
273             return ks
274         else:
275             return ka
276
277     q = lambda x, y: 0  # função q(x)
278     alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solu
        ção do sistema
279
280     L = 1
281     h = L / (n + 1)
282     xi = [i * h for i in range(n + 2)]
283     u_barra_lista = []
284     for x in xi:
285         u_barra_lista.append(u_barra(x, alphas, xi, h))
286         print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f
        }")
287
288     plt.plot(xi, u_barra_lista, label="$d=L/3$, $k_a=0.03$")
289     plt.legend()
290     plt.grid()
291     plt.ylabel("$T(x)$")
292     plt.xlabel("$x$")
293     plt.show()
```

4 Conclusão

4.1 Resultados

Seguem abaixo os resultados e comentários sobre os devidos outputs testados para verificação dos códigos apresentados neste exercício programa.

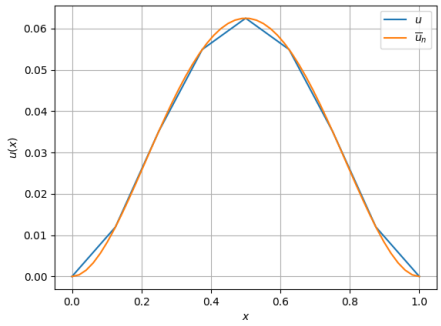
4.1.1 Modo I

Os resultados obtidos para o máximo erro da expressão $\|\bar{u}_n - u\| = \max_{i=1,\ldots,n} |\bar{u}_n(x_i) - u(x_i)|$ para n = 7, 15, 31, 63 são:

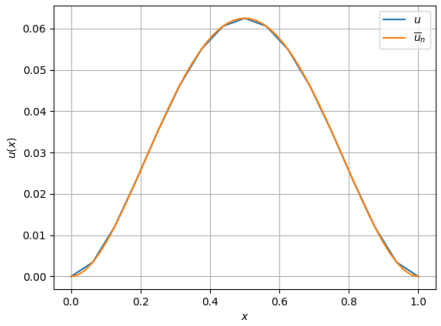
```
Avaliando o máximo erro nos pontos xi para diferentes n's:

n=7 -->
u_barra = 0.000000000000000000000 ; u_exato = 0.000000000000000000000; erro = 0.000000000000000000000

n=15 -->
u_barra = 0.060562133789062402855 5 ; u_exato = 0.060562133789062500000; erro = 0.000000000000000971445

n=31 -->
u_barra = 0.062012672424316052366 4 ; u_exato = 0.062012672424316406250; erro = 0.000000000000003538836

n=63 -->
u_barra = 0.046157836914062090605 3 ; u_exato = 0.046157836914062500000; erro = 0.000000000000004093947
```

Figura 5: Saídas primeiro exercício de validação

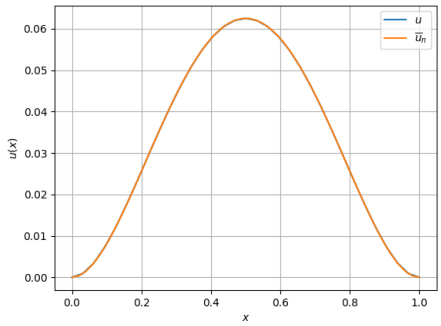
Percebe-se que os resultado difere do esperado, sendo os erros obtidos crescentes conforme n aumenta. Entretanto, criando gráficos dos $\overline{u}_n$ obtidos, percebemos há convergência:



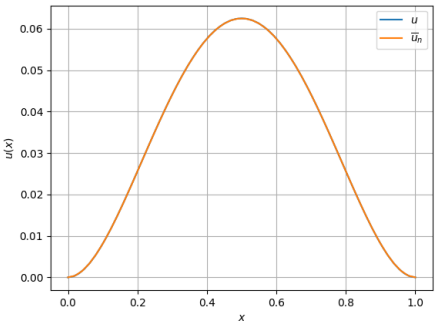
(a) u e $\bar{u}_n$ em função de x para n=7



(b) u e $\bar{u}_n$ em função de x para n=15



(c) u e $\bar{u}_n$ em função de x para n=31



(d) u e $\bar{u}_n$ em função de x para n=63

Figura 6: gráficos de análise de convergência

Assim, podemos analisar pontos que estejam entre os nós x_i para verificar a diminuição do erro. Como exemplo, utilizando os pontos $P_i = 0.33, 0.47, 0.66, 0.72$ os resultados percebe-se a redução do erro com o aumento do n:

```
n=7 -->
x = 0.3300 ; u_barra = 0.047812500000000008327 ; u_exato = 0.048885209999999983870; erro = -0.001072709999999975544
x = 0.4700 ; u_barra = 0.060683593750000000551 ; u_exato = 0.062050810000000048513; erro = -0.001367216250000042962
x = 0.6600 ; u_barra = 0.049394531249999983347 ; u_exato = 0.050355359999999948467; erro = -0.000960828749999965120
x = 0.7200 ; u_barra = 0.039902343750000063838 ; u_exato = 0.040642560000000011858; erro = -0.000740216249999948020

n=15 -->
x = 0.3300 ; u_barra = 0.048614501953124967333 ; u_exato = 0.048885209999999983870; erro = -0.000270708046875016538
x = 0.4700 ; u_barra = 0.061569824218749923004 ; u_exato = 0.062050810000000048513; erro = -0.000480985781250102509
x = 0.6600 ; u_barra = 0.050018310546874906111 ; u_exato = 0.050355359999999948467; erro = -0.000337049453125062356
x = 0.7200 ; u_barra = 0.040437011718749943786 ; u_exato = 0.040642560000000011858; erro = -0.000205548281250057807

n=31 -->
x = 0.3300 ; u_barra = 0.048807296752929399371 ; u_exato = 0.048885209999999983870; erro = -0.000077913247070589899
x = 0.4700 ; u_barra = 0.062032165527343398369 ; u_exato = 0.062050810000000048513; erro = -0.000018644472656606514
x = 0.6600 ; u_barra = 0.050321273803710611649 ; u_exato = 0.050355359999999948467; erro = -0.000034086196289383197
x = 0.7200 ; u_barra = 0.040635681152343484934 ; u_exato = 0.040642560000000011858; erro = -0.000006878847656516251

n=63 -->
x = 0.3300 ; u_barra = 0.048876585960387783358 ; u_exato = 0.048885209999999983870; erro = -0.000008624039612215028
x = 0.4700 ; u_barra = 0.062041897773742395172 ; u_exato = 0.062050810000000048513; erro = -0.000008912226257609678
x = 0.6600 ; u_barra = 0.050340170860290370247 ; u_exato = 0.050355359999999948467; erro = -0.000015189139709624599
x = 0.7200 ; u_barra = 0.040639004707336313926 ; u_exato = 0.040642560000000011858; erro = -0.000003555292663687259
```

Figura 7: Erros para P_i primeiro exercício de validação

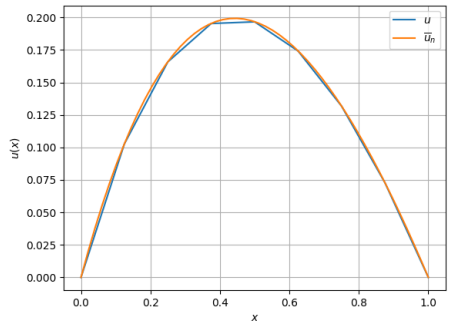
4.1.2 Modo II

Os resultados obtidos para o máximo erro da expressão $\|\bar{u}_n - u\| = \max_{i=1,\ldots,n} |\bar{u}_n(x_i) - u(x_i)|$ para n = 7, 15, 31, 63 são:

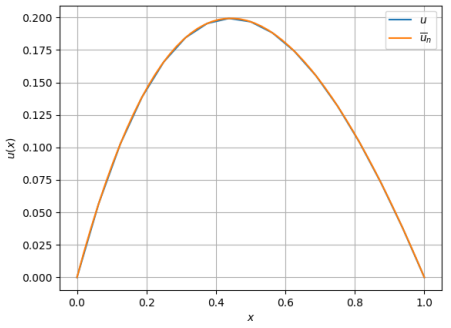
```
Avaliando o máximo erro para diferentes n's:

n=7 -->
u_barra = 0.1953456538986507728950 ; u_exato = 0.1954442007556423388515; erro = 0.000098546856991565965

n=15 -->
u_barra = 0.1992978887681065569559 ; u_exato = 0.1993227038843107257193; erro = 0.000024815116204168763 4

n=31 -->
u_barra = 0.1982210201687935047232 ; u_exato = 0.1982272311448041168802; erro = 0.000006210976010612157 0

n=63 -->
u_barra = 0.1989798148463083615756 ; u_exato = 0.1989813684836631568764; erro = 0.000001553637354795300 9
```

Figura 8: Enunciado primeiro exercício de validação

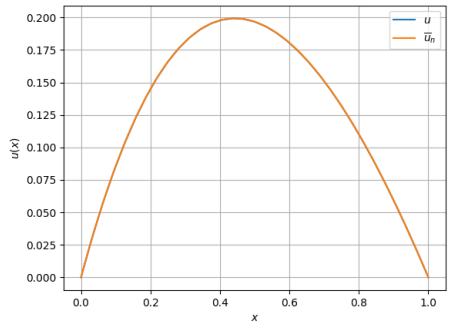
Percebe-se que o erro decai como esperado. O decaimento também é observado nos gráficos gerados:



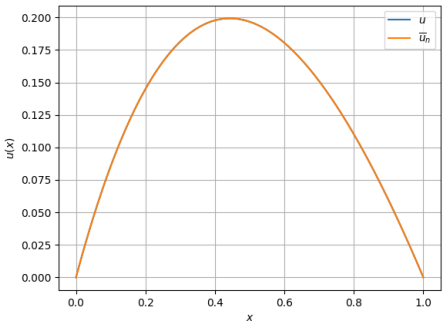
(a) u e $\bar{u}_n$ em função de x para n=7



(b) u e $\bar{u}_n$ em função de x para n=15



(c) u e $\bar{u}_n$ em função de x para n=31



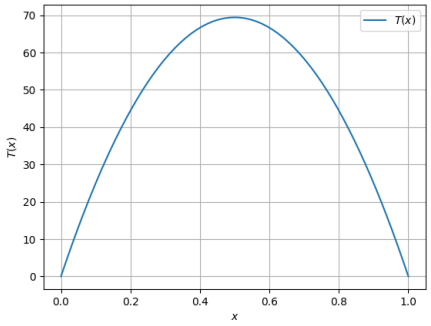
(d) u e $\bar{u}_n$ em função de x para n=63

Figura 9: Análise de convergência

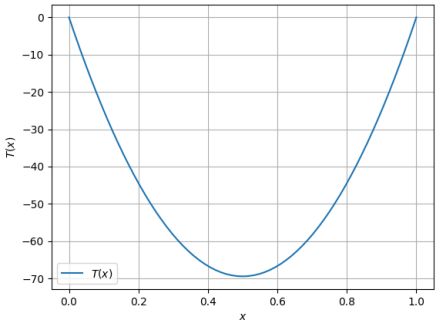
4.1.3 Modo III

Os resultados para a primeira sessão de variações (Q^0 contantes, variando entre Q^0_+ e Q^0_- serem maiores e muito maiores entre si) estão impressos nos gráficos mostrados na Figura 10.

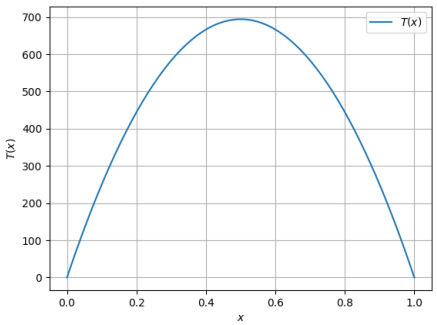
Observa-se que a influência de Q_+ ou Q_- ser muito maior um que o outro afeta apenas a ordem de grandeza da temperatura ao longo do chip. Quando Q_+ é maior, há concentrações de calor no centro do chip e quando o contrário ocorre os pontos de concentração se situam nos extremos do chip.



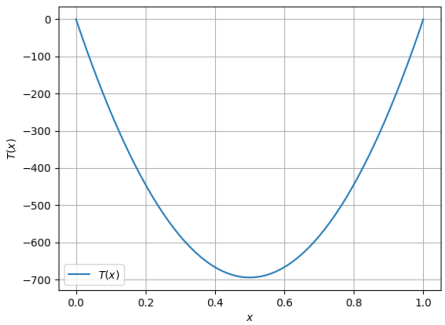
(a) Q_+ maior que Q_-



(b) Q_- maior que Q_+



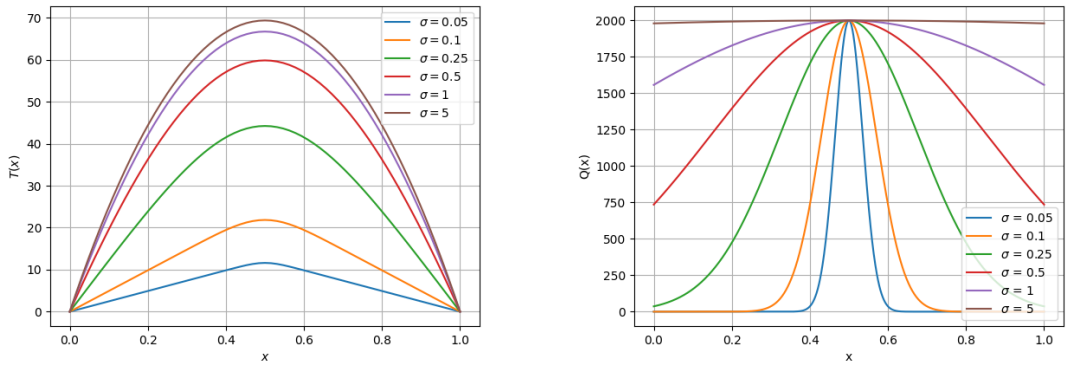
(c) Q_+ muito maior que Q_-



(d) Q_- muito maior que Q_+

Figura 10: Análise da influência de $Q^0_{+/-}$ na distribuição de calor

Quanto à segunda bateria de soluções, gráficos da temperatura e da distribuição de calor ao longo do chip foram utilizados, sendo os gráficos referentes a temperatura tirada do código e os referentes ao calor feitos para a análise do relatório:



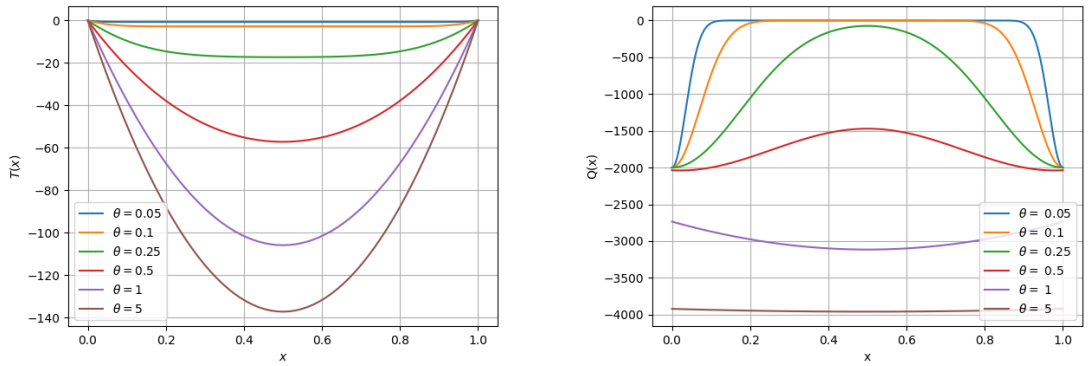
(a) Temperatura em função de x com variações de σ para baixos valores de $Q(x)$

(b) Temperatura em função de x com variações de σ para altos valores de $Q(x)$

Figura 11: Análise da influência de σ na distribuição de temperatura

Analisa-se, primeiramente os casos em que $Q(x) = Q_+(x)$. Pelos resultados obtidos é possível notar que a influência da variação de σ ocorre no quesito de concentração do calor no chip. Para valores muito próximos de zero as temperaturas geradas são menores e com amplitude menores, vale notar que para valores menores que 0.1 a tendência é que a curva de distribuição de calor pelo chip perca o formato de parábola e fique cada vez menor. Para valores maiores a amplitude aumenta e a distribuição assume o formato de parábolas com amplitudes crescentes. Percebe-se, também, que conforme os valores de σ aumentam, as parábolas tendem a convergir a um formato.

Os resultado para a temperatura em função de x são condizentes com o esperado pela análise da distribuição de calor. Para sigmas pequenos, calor fica mais concentrado no centro do chip e para sigmas mais altos a função se aproxima de uma reta, fazendo com que a curva da temperatura se aproxime do caso em que a expressão de Q(x) é constante.

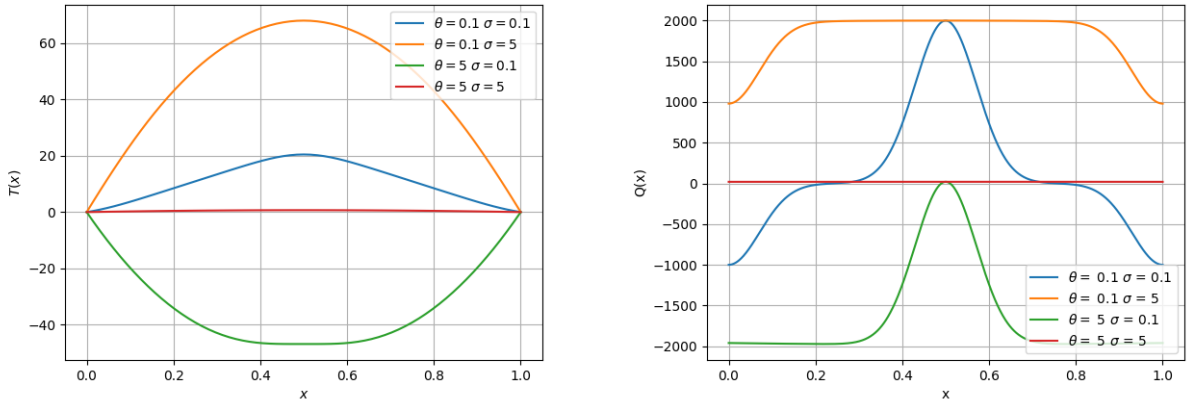


(a) Temperatura em função de x com variações de θ para baixos valores em módulo de $Q(x)$

(b) Temperatura em função de x com variação de θ para altos valores em módulo de $Q(x)$

Figura 12: Análise da influência de θ na distribuição de temperatura

Analisando os caos em que $Q(x) = -Q_{-}(x)$, percebe-se que, ao variar os valores de θ a temperatura cai ao longo do chip, tendo sua curva de variação no formato de uma parábola com concavidade para baixo e com seu mínimo no ponto $\frac{L}{2}$. Ao se reduzir o valor de θ percebe-se o achatamento da curva até o ponto em que esta se aproxima de reta. Comparando, então, com gráfico da distribuição do calor pelo chip, tem-se que que com θ pequeno, o calor se mantém por um maior comprimento em valores próximos de 0, correspondendo assim com o achatamento da parábola no gráfico a). Além disso, semelhante ao caso de σ, com o aumento de θ a curva de calor se aproxima de uma reta de valor constante e, com isso, a curva de temperatura se aproxima do caso em que $Q(x) = -Q_{-}(x)$ com $-Q_{-}(x)$ constante.



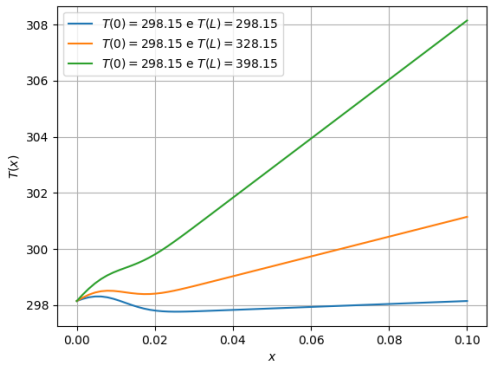
(a) Temperatura em função de x com variação de ambos σ e θ para baixos valores de $Q(x)$

(b) Temperatura em função de x com variações de ambos σ e θ para baixos valores de $Q(x)$

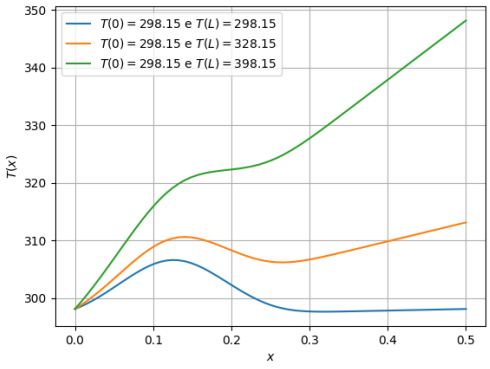
Figura 13: Análise da influência de σ e θ na distribuição de temperatura

Quando analisando o último caso $Q(x) = Q_{+}(x) - Q_{-}(x)$, variando σ e θ repara-se na união dos resultado obtidos nos dois últimos casos. Assim, para $\sigma = 0.1$ e $\theta = 0.1$ temos a curva de distribuição de calor com um pico no centro do chip e com regiões próximas a zero até que se torne menor que zero. Na curva de temperatura tem-se então uma curva com formato semelhante a curva de $\sigma = 0.1$ da Figura 11 (a), mas com o pico levemente achatado. Para $\sigma = 5$ e $\theta = 5$ a curva de distribuição de calor é aproximadamente constante em 0, e, portanto, a temperatura permanece constante e igual a 0. Por fim, as curvas em que $\sigma = 0.1$ e $\theta = 5$; e $\sigma = 5$ e $\theta = 0.1$ tem resultados que mesclam as características de tais casos nas análises anteriores (Figura 11 e Figura 12).

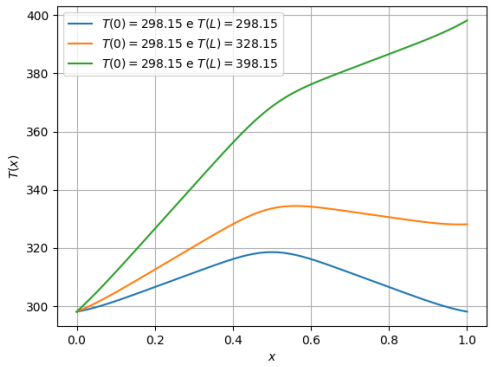
Para a última sessão de variações para analisar a distribuição de calor para condições de contorno não homogêneas:



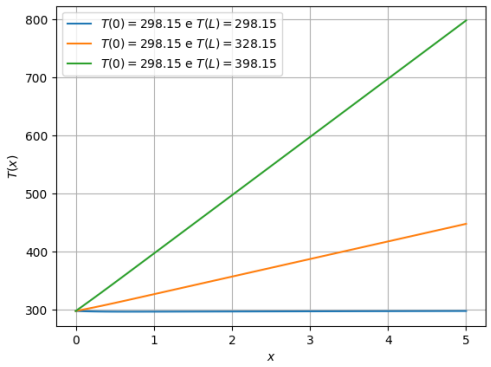
(a) Situação não homogênea com ambos Q não constantes e L=0.1



(b) Situação não homogênea com ambos Q não constantes e L= 0.5



(c) Situação não homogênea com ambos Q não constantes e L=1.0



(d) Situação não homogênea com ambos Q não constantes e L=5.0

Figura 14: gráficos da influência de condições de contorno não homogêneas no comportamento térmico do chip

No quesito de fronteiras não homogêneas, o comportamento é semelhante em quaisquer comprimentos: quanto mais distantes entre si são as temperaturas nos extremos maior o coeficiente linear das curvas de calor e portanto mais rápido a elevação de temperatura. Percebe-se, também que há sempre dois trechos que se aproximam de retas, cada um com inclinações diferentes. Esses trechos são separado pelo ponto de pico de calor, que varia com L. Nos casos $L = 0.1$ e $L = 0.5$ temos que o pico ocorre logo no inicio do chip, para $L = 1$ o pico se encontra exatamente no meio e para $L = 5$ a curva de distribuição de calor não apresenta pico (ou o pico estaria teoricamente fora do chip).

4.1.4 Modo IV

Antes de avaliar a situação proposta no enunciado da tarefa, analisou-se qual a influência da variação do valor de k (constante) na função de temperatura ao longo do chip.

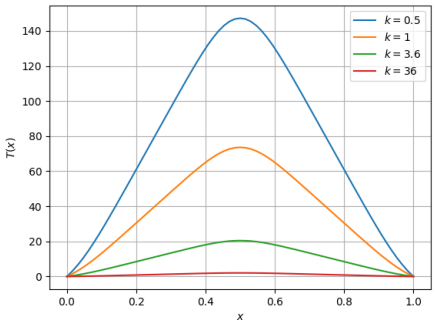
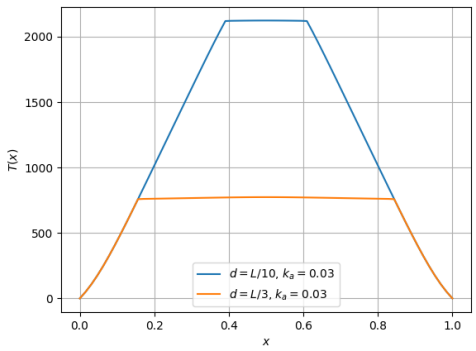


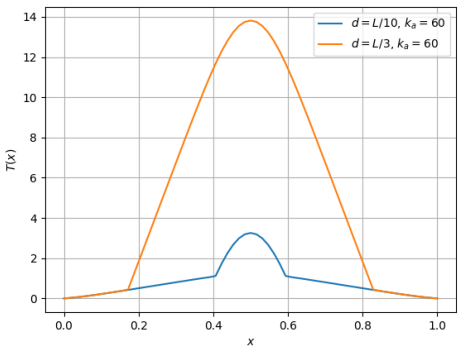
Figura 15: Temperatura ao longo do chip para diferentes valores de k

Dos resultados impressos, é possível concluir que variações em k interferem na temperatura do chip tanto em sua amplitude - valores mais baixos causam máximos maiores- além de como esta se distribui ao longo do chip - valores de k mais altos tendem a dificultar o aumento de temperatura e portanto diminuir suas variações ao longo do chip.

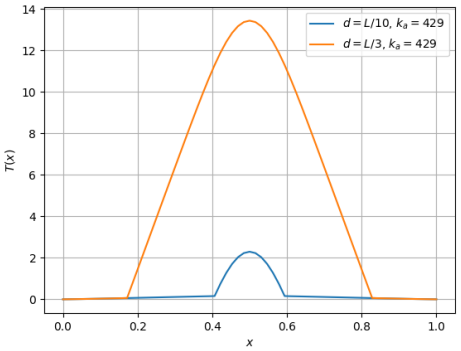
Os resultados para as distribuições de calor para uma situação de material variável em três situações (borda de espuma de poliestireno *(a)*, de alumínio *(b)*, de prata *(c)*) estão apresentados abaixo:



(a) Distribuição de calor em função de x com camada externa de espuma de poliestireno



(b) Distribuição de calor em função de x com camada externa de alumínio



(c) Distribuição de calor em função de x com camada externa de prata

Figura 16: Análise da influência de $k(x)$ de diferentes materiais na distribuição de calor

Pode-se notar que a amplitude da temperatura na parte interna do chip depende diretamente do material que o envolve. Nos casos em que este é inferior à constante de calor do silício (3,6 W /(m K)) , como no caso (a) em que $k_{PE} = 0,03$ W /(m K) a temperatura cresce à valores muito altos até a borda, permitindo que mesmo que com os valores da constante do silício no centro, atingissem valores muito altos. Com o aumento dos valores das constantes dos materiais na borda, essa temperatura diminui cada vez mais até o caso mais extremo de k externo muito maior do que o interno ($k_{prata} = 429W/(mK)$) onde a temperatura se mantém quase estável até o início do interior de silício, causando temperaturas máximas bem menores que nas outras situações.

Vale pontuar ainda, especialmente observando os casos extremos de k muito pequeno ou muito grande, que a espessura da camada de material diferente. Quanto maior a espessura maior a interferência causada pelo k externo: a temperatura máxima aumenta ainda mais para k pequeno e diminui consideravelmente para k maiores.

4.1.5 Conclusões finais

Ao fim deste relatório, conclui-se a série de três exercícios programas propostos pela disciplina. O exercício une os dois últimos exercícios programas em código e em teoria, por meio da aplicação de um importante método de resolução numérica: o método de elementos finitos. O código para a tarefa consta não só na junção do algoritmo para resolução de sistemas de matrizes LU e integrações gaussianas mas também da elaboração de funções para o cálculo das bases ϕ, erros e soluções pelo método de aproximação Ritz-Raleigh e de um espaço vetorial de Splines Lineares. Além disto, subdividiu-se a Main em quatro diferentes modos: dois de validação e dois de análises.

As validações foram concluídas com devido êxito, apresentando os resultados e convergências esperados. E quanto às análises, foram elaboradas as sugeridas no enunciado junto com seus desmembramentos, de autoria própria. Observou-se nos casos das forçantes em equilíbrio a influência da constância dos calores absorvido e retirado , das variáveis σ e θ - responsáveis transformar as funções em não constantes - tanto separadamente quanto em conjunto, além de explorar os efeitos das condições de fronteira não homogêneas na distribuição de temperatura ao longo do chip. Por fim, analisou-se a influência da constante k no comportamento da distribuição de temperatura, assim como o exercício de avaliar a influência de uma camada externa de três materiais diferentes (espuma de poliestireno,alumínio e prata).

Destes teste é possível concluir que um dos maiores desafios para não sobreaquecer o chip residem na produção e absorção de calor não constantes em situações reais, podendo criar concentrações altas no centro da peça, o que, conjuntamente com condições de contorno não homogêneas (tipicamente exposição dos extremos à temperatura ambiente ou de outras peças próximas ao chip) que alteram a distribuição de temperatura para uma distribuição cada vez menos uniforme, chegando quase à um crescimento linear de temperatura. Outro desafio notável é a escolha do material para recobrir o chip, visto que materiais de baixa condutibilidade proporcionam amplitudes de temperaturas muito menores no chip, visto que controlam muito bem o aumento desta ao longo da camada externa e materiais com alta condutibilidade - inclusive o ar atmosférico, cujo valor da constante de condutividade é muito próxima a da espuma de poliestireno utilizada- podem causar não só uma maior amplitude como uma concentração ainda maior de calor no centro do chip.

5 Referências

VALLE,Marcos Eduardo. **Aula 24 Quadratura Gaussiana.**.Departamento de Matemática Aplicada Instituto de Matemática, Estatística e Computação Científica Universidade Estadual de Campinas Disponível em : <https://www.ime.unicamp.br/ valle/Teaching/2015/MS211/Aula24.pdf> Acesso em 31 de maio,2022.

OLIVEIRA,Maria Luísa Bambozzi de. **Integração Numérico**.27 de Outubro, 2010 e 8 de Novembro, 2010. Disponível em

<https://sites.icmc.usp.br/marialuisa/cursos201002/integracao$_numerica.pdf$ > .

Acesso em 30 de maio, 2022.

Hjorth-Jensen,Morten . **Computational Physics Lectures: Numerical integration, from Newton-Cotes quadrature to Gaussian quadrature** . Department of Physics, University of Oslo. 23 de agosto, 2017. Disponível em:

<http://compphysics.github.io/ComputationalPhysics/doc/pub/integrate/html/integrate.html>.

Acesso em: 31 maio 2022.

Gauss quadrature formula.Encyclopedia of Mathematics. Disponível em:

<http://encyclopediaofmath.org/index.php?title=Gauss$_quadrature_formulaoldid$ = 43647 > .

Acesso em: 30 maio 2022.

INTEGRAÇÃO NUMÉRICA - QUADRATURA DE GAUSS LEGENDRE.Métodos Numéricos. Youtube. 21 de novembro de 2020. 22:33. Disponível em: <https://www.youtube.com/watch?v=6W5Y2JWQnUg>. Acesso em: 31 de Maio de 2022.

Calculadora de Integrais Duplas - Wolfram Alpha. Disponível em:

<https://www.wolframalpha.com/input?i=double+integral>. Acesso em: 2 junho 2022.

NumPy documentation — NumPy v1.22 Manual. Numpy.org. Disponível em: <https://numpy.org/doc/stable/index.html>. Acesso em: 2 maio 2022.

Quadratura Gaussiana + Algoritmo em Python.Produção: Sidnei fc. Youtube. 2021. 27:49. Disponível em : <https://www.youtube.com/watch?v=bu8trr9Qm1Y> Acesso em 01/06/2022

JUSTINO. Lucas; **Método da Decomposição LU para a solução numérica de equações lineares**. Disponível em :

"http://www.facom.ufu.br/ dino/disciplinas/GBC051/Decomp_LU_Lucas.pdf"

ECT / UFRN. **Decomposição LU (Lower Upper)**. Disponível em:

<https://cn.ect.ufrn.br/index.php?r=conteudo%2Fsislin-lu>. Acesso em: 1 maio 2022.

RISTO HINNO. LU decomposition - Risto Hinno - Medium. Medium. Disponível em: <https://ristohinno.medium.com/lu-decomposition-41a3cb0d1ba>. Acesso em: 1 maio 2022.

Matrix Calculator. Matrixcalc.org. Disponível em: <https://matrixcalc.org/pt/slu.html>. Acesso em: 2 maio 2022.

Calculadora de Decomposição LU - eMathHelp. Emathhelp.net. Disponível em: <https://www.emathhelp.net/pt/calculators/linear-algebra/lu-decomposition-calculator/>. Acesso em: 2 maio 2022.

Finite Element Method.Produção: Julian Roth. Youtube.2021. 32:18. Disponível em : <https://www.youtube.com/watch?v=P4lBRuY7pC4> Acesso em: 5 de julho 2022.

Oficina MEF no Python (aula 3). Produção : LabMA UFRRJ. Youtube.1:10:10. Disponível em: <https://www.youtube.com/watch?v=qDFGgMPbVQQ>

A Apêndices

A.1 functions.py

```
1  import numpy as np
2
3  # Resolve sistema tridiagonais
4  def sistemaTridiagLU(a, b, c, d, n):
5      """
6      a: vetor de coeficientes da diagonal secundaria inferior
7      b: vetor de coeficientes da diagonal principal
8      c: vetor de coeficientes da diagonal secundaria superior
9      d: vetor resultado
10     n: dimensão do sistema nxn
11
12     """
13     # Calcula vetores u e l
14     u = [b[0]]
15     l = []
16     for i in range(1, n):
17         l.append(a[i] / u[i - 1])
18         u.append(b[i] - l[i - 1] * c[i - 1])
19
20     # Calcula solução de L*y = d
21     y = [d[0]]
22     for i in range(1, n):
23         y.append(d[i] - l[i - 1] * y[i - 1])
24
25     # Calcula solução de U*x = y
26     x = [0] * n
27     x[n - 1] = y[n - 1] / u[n - 1]
28     for i in reversed(range(0, n - 1)):
29         x[i] = (y[i] - c[i] * x[i + 1]) / u[i]
30
31     return x
32
33
34 # Calculo da integral simples ou dupla
35 def integral_simples_ou_dupla(
36     f, a, b, pontos_e_pesos_x, c=lambda x: 0, d=lambda x: 1,
   pontos_e_pesos_y=None
37 ):
38     """
39     f: função de x e y a ser integrada
40     a: limite inferior do intervalo de integração em x
41     b: limite superior do intervalo de integração em x
42     pontos_e_pesos_x: pontos e pesos para o método de Gauss em x (
   deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
43     c: limite inferior do intervalo de integração em y (função de x)
44     d: limite superior do intervalo de integração em y (função de x)
```

```
45      pontos_e_pesos_y: pontos e pesos para o método de Gauss em y (
        deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
46
47      """
48
49      pontos_e_pesos_y = (
50          pontos_e_pesos_x if pontos_e_pesos_y is None else
        pontos_e_pesos_y
51      )
52
53      I = 0
54      for x_i, w_i in pontos_e_pesos_x:
55          # troca de variavel de x para o intervalo [a,b]
56          x_i = (a + b) / 2 + (b - a) * x_i / 2
57          F = 0
58          for y_ij, w_ij in pontos_e_pesos_y:
59              # troca de variavel de y para o intervalo [c(x_i),d(x_i)
        ]
60              y_ij = (c(x_i) + d(x_i)) / 2 + (d(x_i) - c(x_i)) * y_ij
        / 2
61              F += w_ij * f(x_i, y_ij) * (d(x_i) - c(x_i)) / 2
62          I += w_i * F * (b - a) / 2
63
64      return I
65
66
67  def calcula_produto_interno_phis_diagonal_principal(
68      k, q, past_xi, current_xi, next_xi, h, pontos_e_pesos_phi
69  ):
70      """
71      k: função k(x)
72      q: função q(x)
73      past_xi: xi[i-1], xi anterior
74      current_xi: xi[i], xi atual
75      next_xi: xi[i+1], xi posterior
76      h: tamanho do intervalo
77      pontos_e_pesos_phi: pontos e pesos para o método de Gauss em phi
        (deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
78
79      """
80
81      # Calcula o produto interno de phii e phii
82      phii_phii_1 = lambda x, y: k(x, y) + (x - past_xi) ** 2 * q(x, y
        )
83      phii_phii_2 = lambda x, y: k(x, y) + (next_xi - x) ** 2 * q(x, y
        )
84      return (1 / h) ** 2 * (
85          integral_simples_ou_dupla(
86              phii_phii_1, past_xi, current_xi, pontos_e_pesos_phi
87          )  # intervalo de integração [past_xi, current_xi]
88          + integral_simples_ou_dupla(
```

```
            phii_phii_2, current_xi, next_xi, pontos_e_pesos_phi
        )  # intervalo de integração [current_xi, next_xi]
    )


def calcula_produto_interno_phis_diagonal_secundaria_inferior(
    k, q, past_xi, current_xi, h, pontos_e_pesos_phi
):
    """
    k: função k(x)
    q: função q(x)
    past_xi: xi[i-1], xi anterior
    current_xi: xi[i], xi atual
    h: tamanho do intervalo
    pontos_e_pesos_phi: pontos e pesos para o método de Gauss em phi
    (deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
    """

    # Calcula o produto interno de phii e phii+1
    phii_phij = lambda x, y: k(x, y) + (current_xi - x) * (x -
   past_xi) * q(x, y)
    return -((1 / h) ** 2) * integral_simples_ou_dupla(
        phii_phij, past_xi, current_xi, pontos_e_pesos_phi
    )  # intervalo de integração [past_xi, current_xi]


def calcula_produto_interno_phis_diagonal_secundaria_superior(
    k, q, current_xi, next_xi, h, pontos_e_pesos_phi
):
    """
    k: função k(x)
    q: função q(x)
    current_xi: xi[i], xi atual
    next_xi: xi[i+1], xi posterior
    h: tamanho do intervalo
    pontos_e_pesos_phi: pontos e pesos para o método de Gauss em phi
    (deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
    """

    # Calcula o produto interno de phii e phii-1
    phij_phii = lambda x, y: k(x, y) + (next_xi - x) * (x -
   current_xi) * q(x, y)
    return -((1 / h) ** 2) * integral_simples_ou_dupla(
        phij_phii, current_xi, next_xi, pontos_e_pesos_phi
    )  # intervalo de integração [current_xi, next_xi]


def calcula_produto_interno_f_phi(
    f, past_xi, current_xi, next_xi, h, pontos_e_pesos_phi
):
    """
```

```
136     f: função f(x)
137     past_xi: xi[i-1], xi anterior
138     current_xi: xi[i], xi atual
139     next_xi: xi[i+1], xi posterior
140     h: tamanho do intervalo
141     pontos_e_pesos_phi: pontos e pesos para o método de Gauss em phi
        (deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
142     """
143
144     # Calcula o produto interno de f(x) e phii
145     f_phii1 = lambda x, y: (x - past_xi) * f(x, y)
146     f_phii2 = lambda x, y: (next_xi - x) * f(x, y)
147     return (
148         1
149         / h
150         * (
151             integral_simples_ou_dupla(f_phii1, past_xi, current_xi,
    pontos_e_pesos_phi)
152             + integral_simples_ou_dupla(
153                 f_phii2, current_xi, next_xi, pontos_e_pesos_phi
154             )
155         )
156     )
157
158
159 def sol_sistema_linear_tridiagonal(f, k, q, n, pontos_e_pesos_phi, L
    =1):
160     """
161     f: função f(x)
162     k: função k(x)
163     q: função q(x)
164     n: número de pontos
165     pontos_e_pesos_phi: pontos e pesos para o método de Gauss em phi
        (deve ser da forma [(ponto1, peso1), (ponto2, peso2), ...].)
166     L: opcional, define interval [0,L]
167     """
168
169     h = L / (n + 1)
170     xi = [i * h for i in range(n + 2)]
171
172     # Vetores do sistema linear (tridigonal)
173     b = []  # diagonal principal
174     a = []  # diagonal secundária inferior
175     c = []  # diagonal secundária superior
176     d = []  # vetor de termos independentes
177     for i in range(n):
178         past_xi = xi[i]
179         current_xi = xi[i + 1]
180         next_xi = xi[i + 2]
181
182         b.append(
```

```
183             calcula_produto_interno_phis_diagonal_principal(
184                 k, q, past_xi, current_xi, next_xi, h,
    pontos_e_pesos_phi
185             )
186         )
187         a.append(
188
    calcula_produto_interno_phis_diagonal_secundaria_inferior(
189                 k, q, past_xi, current_xi, h, pontos_e_pesos_phi
190             )
191         )
192         c.append(
193
    calcula_produto_interno_phis_diagonal_secundaria_superior(
194                 k, q, current_xi, next_xi, h, pontos_e_pesos_phi
195             )
196         )
197         d.append(
198             calcula_produto_interno_f_phi(
199                 f, past_xi, current_xi, next_xi, h,
    pontos_e_pesos_phi
200             )
201         )
202
203     # Resolve o sistema linear
204     alphas = np.array(sistemaTridiagLU(a, b, c, d, n))
205
206     return alphas
207
208
209 def phi(x, xi, i, h):
210     """
211     x: valor de x
212     xi: vetor de pontos
213     i: índice do ponto de xi
214     h: tamanho do intervalo
215     """
216
217     if xi[i - 1] < x <= xi[i]:
218         return (x - xi[i - 1]) / h
219     elif xi[i] < x <= xi[i + 1]:
220         return (xi[i + 1] - x) / h
221     else:
222         return 0
223
224
225 def u_barra(x, alphas, xi, h):
226     """
227     x: valor de x
228     alphas: vetor de alphas
229     xi: vetor de pontos
```

```
    h: tamanho do intervalo

    """

    u_barra = 0
    for i in range(len(alphas)):
        u_barra += alphas[i] * phi(x, xi, i + 1, h)
    return u_barra


def u_barra_nao_homogenea(x, a, b, alphas, xi, h, L=1):
    """
    x: valor de x
    a: valor de u(0)
    b: valor de u(L)
    alphas: vetor de alphas
    xi: vetor de pontos
    h: tamanho do intervalo

    """
    return u_barra(x, alphas, xi, h) + (a + (b - a) * x) / L


def maior_erro(n, u_exato, alphas, L=1):
    """
    n: número de pontos
    u_exato: função u(x)
    alphas: vetor de alphas
    L: opcional, define interval [0,L]
    """

    h = L / (n + 1)
    xi = [i * h for i in range(n + 2)]
    erro_max = 0
    u_barra_erro_max = 0
    u_exato_erro_max = 0
    for x in xi:
        if abs(u_exato(x) - u_barra(x, alphas, xi, h)) > erro_max:
            erro_max = abs(u_exato(x) - u_barra(x, alphas, xi, h))
            u_barra_erro_max = u_barra(x, alphas, xi, h)
            u_exato_erro_max = u_exato(x)
    return erro_max, u_barra_erro_max, u_exato_erro_max
```

A.2 Main.py

```
"""
EP3 - Modelagem de um Sistema de Resfriamento de Chips

Nome: Laura do Prado Gonçalves Pinto
NUSP: 11819960
```

```
6
7  Nome: Mateus Stano Junqueira
8  NUSP: 11804845
9  """
10
11 from functions import *
12 import numpy as np
13 import matplotlib.pyplot as plt
14
15 # pontos e pesos pré estabelecidos
16 # na forma de n = [(x_j,w_j) ,...]
17 n6 = [
18     (0.23861918608319690863050172, 0.46791393457269104738987034),
19     (0.66120938646626451366139960, 0.36076157304813860756983351),
20     (0.93246951420315202781230155, 0.17132449237917034504029614),
21     (-0.23861918608319690863050172, 0.46791393457269104738987034),
22     (-0.66120938646626451366139960, 0.36076157304813860756983351),
23     (-0.93246951420315202781230155, 0.17132449237917034504029614),
24 ]
25 n8 = [
26     (0.18343464249564980493947614, 0.36268378337836198296515042),
27     (0.52553240991632898581773904, 0.31370664587788728733796220),
28     (0.79666477413626739591553914, 0.22238103445337447054435604),
29     (0.96028985649753623168356091, 0.10122853629037625915253140),
30     (-0.18343464249564980493947614, 0.36268378337836198296515042),
31     (-0.52553240991632898581773904, 0.31370664587788728733796220),
32     (-0.79666477413626739591553914, 0.22238103445337447054435604),
33     (-0.96028985649753623168356091, 0.10122853629037625915253140),
34 ]
35 n10 = [
36     (0.14887433898163121088482605, 0.29552422471475287017389303),
37     (0.43339539412924719079926593, 0.26926671930999635509122690),
38     (0.67940956829902440623432736, 0.21908636251598204399553493),
39     (0.86506336668898451073209669, 0.14945134915058059314577630),
40     (0.97390652851717172007796401, 0.06667134430868813759356883),
41     (-0.14887433898163121088482605, 0.29552422471475287017389303),
42     (-0.43339539412924719079926593, 0.26926671930999635509122690),
43     (-0.67940956829902440623432736, 0.21908636251598204399553493),
44     (-0.86506336668898451073209669, 0.14945134915058059314577630),
45     (-0.97390652851717172007796401, 0.06667134430868813759356883),
46 ]
47
48
49 def main():
50     print("Escolha um modo:")
51     print("Modo 1 - Validação no intervalo [0,1] com k(x)=1, q(x)=0,
       f(x)=12x(1-x)-2")
52     print("Modo 2 - Validação no intervalo [0,1] com k(x)=e**x, q(x)
       =0, f(x)=e**x+1")
53     print("Modo 3 - Equilíbrio com forçantes de calor")
54     print("Modo 4 - Equilibrio com variacao de material")
```

```
55      modo = int(input("Modo: "))
56
57      if modo == 1:
58          print(
59              "\nModo 1 - Validação no intervalo [0,1] com k(x)=1, q(x
    )=0, f(x)=12x(1-x)-2"
60          )
61          print("\nSolução exata u(x) = (x**2 *(1 - x)**2")
62          print("\nParametros: \nk(x)=1, \nq(x)=0, \nf(x)=12x(1-x)-2")
63
64          input("\n\nAvaliando o máximo erro nos pontos xi para
    diferentes n's:")
65
66          for n in [7, 15, 31, 63]:  # números de pontos
67
68              print(f"\nn={n} -->")
69
70              f = lambda x, y: 12 * x * (1 - x) - 2  # função f(x)
71              k = lambda x, y: 1  # função k(x)
72              q = lambda x, y: 0  # função q(x)
73              u_exato = lambda x: x**2 * (1 - x) ** 2  # solução exata
74              alphas = sol_sistema_linear_tridiagonal(
75                  f, k, q, n, n10
76              )  # solução do sistema
77
78              erro_max, u_barra_erro_max, u_exato_erro_max =
    maior_erro(
79                  n, u_exato, alphas
80              )
81              print(
82                  f"u_barra = {u_barra_erro_max:.22f} ; u_exato = {
    u_exato_erro_max:.22f}; erro = {erro_max:.22f}"
83              )
84
85          input("\nPressione Enter para ver todos os erros calculados.
    ")
86          for n in [7, 15, 31, 63]:
87
88              print(f"\nn={n} -->")
89
90              alphas = sol_sistema_linear_tridiagonal(
91                  f, k, q, n, n10
92              )  # solução do sistema
93              L = 1
94              h = L / (n + 1)
95              xi = [i * h for i in range(n + 2)]
96              u_barra_lista = []
97              for x in xi:
98                  u_barra_lista.append(u_barra(x, alphas, xi, h))
99                  print(
```

```
100                     f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f} ; u_exato = {u_exato(x):.22f}; erro = {u_barra(x,alphas,xi,h)-u_exato(x):.22f}"
101                 )
102             u_exato_lista = [u_exato(x) for x in np.linspace(0, 1)]
103 
104             plt.plot(xi, u_barra_lista, label="$u$")
105             plt.plot(np.linspace(0, 1), u_exato_lista, label="$\overline{u}_n$")
106             plt.legend()
107             plt.grid()
108             plt.ylabel("$u(x)$")
109             plt.xlabel("$x$")
110             plt.show()
111 
112         input("\n\nAvaliando o erro nos pontos p=(x[i-1]-xi)/2 para diferentes n's:")
113 
114         for n in [7, 15, 31, 63]:  # números de pontos
115 
116             print(f"\nn={n} -->")
117 
118             f = lambda x, y: 12 * x * (1 - x) - 2  # função f(x)
119             k = lambda x, y: 1  # função k(x)
120             q = lambda x, y: 0  # função q(x)
121             u_exato = lambda x: x**2 * (1 - x) ** 2  # solução exata
122             alphas = sol_sistema_linear_tridiagonal(
123                 f, k, q, n, n10
124             )  # solução do sistema
125 
126             h = L / (n + 1)
127             xi = [i * h for i in range(n + 2)]
128 
129             # testanto em nos pontos pi
130             P_i = [0.33, 0.47, 0.66, 0.72]
131             for x in P_i:
132                 print(
133                     f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f} ; u_exato = {u_exato(x):.22f}; erro = {u_barra(x,alphas,xi,h)-u_exato(x):.22f}"
134                 )
135     elif modo == 2:
136         print(
137             "\nModo 2 - Validação no intervalo [0,1] com k(x)=e**x, q(x)=0, f(x)=e**(x)+1"
138         )
139         print("\nSolução exata u(x) = (x-1)*(e**(-x) - 1)")
140         print("\nParametros: \nk(x)=1, \nq(x)=0, \nf(x)=e**(x)+1")
141 
142         input("\n\nAvaliando o máximo erro para diferentes n's:")
143 
```

```
144         for n in [7, 15, 31, 63]:  # números de pontos
145
146             print(f"\nn={n} —>")
147
148             f = lambda x, y: np.exp(x) + 1  # função f(x)
149             k = k = lambda x, y: np.exp(x)  # função k(x)
150             q = lambda x, y: 0  # função q(x)
151             u_exato = lambda x: (x - 1) * (np.exp(-x) - 1)  # soluçã
    o exata
152             alphas = sol_sistema_linear_tridiagonal(
153                 f, k, q, n, n10
154             )  # solução do sistema
155
156             erro_max, u_barra_erro_max, u_exato_erro_max =
    maior_erro(
157                 n, u_exato, alphas
158             )
159             print(
160                 f"u_barra = {u_barra_erro_max:.22f} ; u_exato = {
    u_exato_erro_max:.22f}; erro = {erro_max:.22f}"
161             )
162
163         input("\nPressione Enter para ver todos os erros calculados.
    ")
164         for n in [7, 15, 31, 63]:
165
166             print(f"\nn={n} —>")
167
168             alphas = sol_sistema_linear_tridiagonal(
169                 f, k, q, n, n10
170             )  # solução do sistema
171             L = 1
172             h = L / (n + 1)
173             xi = [i * h for i in range(n + 2)]
174             u_barra_lista = []
175             for x in xi:
176                 u_barra_lista.append(u_barra(x, alphas, xi, h))
177                 print(
178                     f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h
    ):.22f} ; u_exato = {u_exato(x):.22f}; erro = {u_barra(x,alphas,
    xi,h)-u_exato(x):.22f}"
179                 )
180             u_exato_lista = [u_exato(x) for x in np.linspace(0, 1)]
181
182             plt.plot(xi, u_barra_lista, label="$u$")
183             plt.plot(np.linspace(0, 1), u_exato_lista, label="$\
    overline{u}_n$")
184             plt.legend()
185             plt.grid()
186             plt.ylabel("$u(x)$")
187             plt.xlabel("$x$")
```

```
188             plt.show()
189
190     elif modo == 3:
191         print("\nModo 3 - Equilíbrio com forçantes de calor")
192         print("\nParametros: \nk(x)=3.6, \nq(x)=0, \nf(x)=Q(x)")
193         print(
194             "\nSendo Q(x) representa a soma do calor gerado pelo
    chip (Q+) e o calor retirado pelo resfriador (Q-)"
195         )
196         print("\n\nVariando parametros:")
197
198         print(
199             "\n  Q+ e Q- constantes e Q+ > Q- (portanto Q(x)
    constante e maior que zero)"
200         )
201         input(" Parametros: Q+ - Q- = 2000")
202
203         n = 63
204         f = lambda x, y: 2000  # função f(x)
205         k = lambda x, y: 3.6  # função k(x)
206         q = lambda x, y: 0  # função q(x)
207         alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  #
    solução do sistema
208
209         L = 1
210         h = L / (n + 1)
211         xi = [i * h for i in range(n + 2)]
212         u_barra_lista = []
213         for x in xi:
214             u_barra_lista.append(u_barra(x, alphas, xi, h))
215             print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f}")
216
217         plt.plot(xi, u_barra_lista, label="$T(x)$")
218         plt.legend()
219         plt.grid()
220         plt.ylabel("$T(x)$")
221         plt.xlabel("$x$")
222         plt.show()
223
224         print("\n\n  Q+ e Q- constantes e Q+ >>> Q-")
225         input(" Parametros: Q+ - Q- = 20000")
226
227         n = 63
228         f = lambda x, y: 20000  # função f(x)
229         k = lambda x, y: 3.6  # função k(x)
230         q = lambda x, y: 0  # função q(x)
231         alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  #
    solução do sistema
232
233         L = 1
```

```
234         h = L / (n + 1)
235         xi = [i * h for i in range(n + 2)]
236         u_barra_lista = []
237         for x in xi:
238             u_barra_lista.append(u_barra(x, alphas, xi, h))
239             print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f}")
240 
241         plt.plot(xi, u_barra_lista, label="$T(x)$")
242         plt.legend()
243         plt.grid()
244         plt.ylabel("$T(x)$")
245         plt.xlabel("$x$")
246         plt.show()
247 
248         print(
249             "\n\n  Q+ e Q- constantes e Q+ < Q- (portanto Q(x)
    constante e menor que zero)"
250         )
251         input(" Parametros: Q+ - Q- = -2000")
252 
253         n = 63
254         f = lambda x, y: -2000  # função f(x)
255         k = lambda x, y: 3.6  # função k(x)
256         q = lambda x, y: 0  # função q(x)
257         alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  #
    solução do sistema
258 
259         L = 1
260         h = L / (n + 1)
261         xi = [i * h for i in range(n + 2)]
262         u_barra_lista = []
263         for x in xi:
264             u_barra_lista.append(u_barra(x, alphas, xi, h))
265             print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f}")
266 
267         plt.plot(xi, u_barra_lista, label="$T(x)$")
268         plt.legend()
269         plt.grid()
270         plt.ylabel("$T(x)$")
271         plt.xlabel("$x$")
272         plt.show()
273 
274         print("\n\n  Q+ e Q- constantes e Q+ <<< Q-")
275         input(" Parametros: Q+ - Q- = -20000")
276 
277         n = 63
278         f = lambda x, y: -20000  # função f(x)
279         k = lambda x, y: 3.6  # função k(x)
280         q = lambda x, y: 0  # função q(x)
```

```
281         alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  #
    solução do sistema
282 
283         L = 1
284         h = L / (n + 1)
285         xi = [i * h for i in range(n + 2)]
286         u_barra_lista = []
287         for x in xi:
288             u_barra_lista.append(u_barra(x, alphas, xi, h))
289             print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f}")
290 
291         plt.plot(xi, u_barra_lista, label="$T(x)$")
292         plt.legend()
293         plt.grid()
294         plt.ylabel("$T(x)$")
295         plt.xlabel("$x$")
296         plt.show()
297 
298         print("\n\n  Q+ modelado por Q+(x) = Q+0 * e**-(x-L/2)**2/
    sigma**2 e Q-=0.")
299         print("  Variando sigma")
300         input("\n  Parametros: Q+0 = 2000, L=1, sigma = variando, Q
    -0 = 0")
301         n = 63
302         Q0 = 2000
303         sigmas = [0.05, 0.1, 0.25, 0.5, 1, 5]
304         L = 1
305         Q_0 = 0
306         for sigma in sigmas:
307             f = (
308                 lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) /
    sigma**2) - Q_0
309             )  # função f(x)
310             k = lambda x, y: 3.6  # função k(x)
311             q = lambda x, y: 0  # função q(x)
312             alphas = sol_sistema_linear_tridiagonal(
313                 f, k, q, n, n10
314             )  # solução do sistema
315 
316             h = L / (n + 1)
317             xi = [i * h for i in range(n + 2)]
318             u_barra_lista = []
319             for x in xi:
320                 u_barra_lista.append(u_barra(x, alphas, xi, h))
321                 print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi
    ,h):.22f}")
322 
323             plt.plot(xi, u_barra_lista, label="$\sigma=$" + str(
    sigma))
324         plt.legend()
```

```
325         plt.grid()
326         plt.ylabel("$T(x)$")
327         plt.xlabel("$x$")
328         plt.show()
329 
330         print(
331             "\n\n  Q+=0 e Q- modelado por Q-(x) = Q-0 * (e**(-x**2/
    **2) + e**(-(x-L)**2/  **2))"
332         )
333         print("Variando theta")
334         input("\n  Parametros: Q+0 = 0, L=1, theta = variando, Q-0 =
    2000")
335 
336         n = 63
337         Q0 = 0
338         thetas = [0.05, 0.1, 0.25, 0.5, 1, 5]
339         L = 1
340         Q_0 = 2000
341         for theta in thetas:
342             f = lambda x, y: Q0 - (
343                 Q_0
344                 * (
345                     np.e ** (-(x**2) / theta**2)
346                     + np.e ** (-((x - L) ** 2) / theta**2)
347                 )
348             )  # função f(x)
349             k = lambda x, y: 3.6  # função k(x)
350             q = lambda x, y: 0  # função q(x)
351             alphas = sol_sistema_linear_tridiagonal(
352                 f, k, q, n, n10
353             )  # solução do sistema
354 
355             L = 1
356             h = L / (n + 1)
357             xi = [i * h for i in range(n + 2)]
358             u_barra_lista = []
359             for x in xi:
360                 u_barra_lista.append(u_barra(x, alphas, xi, h))
361                 print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi
    ,h):.22f}")
362 
363             plt.plot(xi, u_barra_lista, label="$\u03B8=$" + str(
    theta))
364         plt.legend()
365         plt.grid()
366         plt.ylabel("$T(x)$")
367         plt.xlabel("$x$")
368         plt.show()
369 
370         print(
```

```
371             "\n\n  Q+ modelado por Q+(x) = Q+0 * e**-(x-L/2)**2/
    sigma**2 e Q- modelado por Q-(x) = Q-0 * (e**(-x**2/  **2) + e
    **(-(x-L)**2/  /**2))."
372         )
373         print("Variando sigma e theta.")
374         input(
375             "\n  Parametros: Q+0 = 2000, L=1, sigma = variando,
    theta = variando, Q-0 = 1000"
376         )
377
378         n = 63
379         L = 1
380         Q0 = 2000
381         thetas = [0.1, 5]
382         sigmas = [0.1, 5]
383         L = 1
384         Q_0 = 1000
385         for theta in thetas:
386             for sigma in sigmas:
387                 f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) /
     sigma**2) - (
388                     Q_0
389                     * (
390                         np.e ** (-(x**2) / theta**2)
391                         + np.e ** (-((x - L) ** 2) / theta**2)
392                     )
393                 )  # função f(x)
394                 k = lambda x, y: 3.6  # função k(x)
395                 q = lambda x, y: 0  # função q(x)
396                 alphas = sol_sistema_linear_tridiagonal(
397                     f, k, q, n, n10
398                 )  # solução do sistema
399
400                 h = L / (n + 1)
401                 xi = [i * h for i in range(n + 2)]
402                 u_barra_lista = []
403                 for x in xi:
404                     u_barra_lista.append(u_barra(x, alphas, xi, h))
405                     print(f"x = {x:.4f} ; u_barra = {u_barra(x,
    alphas,xi,h):.22f}")
406
407                 plt.plot(
408                     xi,
409                     u_barra_lista,
410                     label="$\u03B8=$" + str(theta) + " $\sigma=$" +
    str(sigma),
411                 )
412         plt.legend()
413         plt.grid()
414         plt.ylabel("$T(x)$")
415         plt.xlabel("$x$")
```

```
416         plt.show()
417
418         print(
419             "\n\n  Q+ modelado por Q+(x) = Q+0 * e**-(x-L/2)**2/
    sigma**2 e Q- modelado por Q-(x) = Q-0 * (e**(-x**2/  **2) + e
    **(-(x-L)**2/  /**2))."
420         )
421         print("Variando condições de contorno e L")
422         input(
423             "\n  Parametros: Q+0 = 2000, L=1, sigma = 0.1, theta =
    0.1, Q-0 = 1000, T(0)=298.15, T(L) = 389.15"
424         )
425
426         T0 = 298.15
427         T1s = [298.15, 328.15, 398.15]
428         n = 63
429         Ls = [0.1, 0.5, 1, 5]
430         Q0 = 2000
431         theta = 0.1
432         sigma = 0.1
433         Q_0 = 1000
434         for L in Ls:
435             f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) /
    sigma**2) - (
436                 Q_0
437                 * (
438                     np.e ** (-(x**2) / theta**2)
439                     + np.e ** (-((x - L) ** 2) / theta**2)
440                 )
441             )  # função f(x)
442             k = lambda x, y: 3.6  # função k(x)
443             q = lambda x, y: 0  # função q(x)
444             alphas = sol_sistema_linear_tridiagonal(
445                 f, k, q, n, n10
446             )  # solução do sistema
447
448             h = L / (n + 1)
449             xi = [i * h for i in range(n + 2)]
450             for T1 in T1s:
451                 u_barra_lista = []
452                 for x in xi:
453                     u_barra_lista.append(
454                         u_barra_nao_homogenea(x, T0, T1, alphas, xi,
     h)
455                     )
456                     print(
457                         f"x = {x:.4f} ; u_barra = {
    u_barra_nao_homogenea(x,T0,T1,alphas,xi,h):.22f}"
458                     )
459
460                 plt.plot(
```

```
461                     xi,
462                     u_barra_lista,
463                     label="$T(0)=$" + str(T0) + " e $T(L)=$" + str(
    T1),
464                 )
465             plt.legend()
466             plt.grid()
467             plt.ylabel("$T(x)$")
468             plt.xlabel("$x$")
469             plt.show()
470
471     elif modo == 4:
472         print("\nModo 4 - Equilibrio com variacao de material")
473         print("\nAnalisando caso em que k(x)=k com k variando")
474         input("\nParametros: \n L=1, q(x)=0, theta=1, sigma=1, Q
    +0=2,Q-0=2 f(x)=Q(x)")
475
476         n = 63
477         ks = [0.5, 1, 3.6, 36]
478         Q0 = 2000
479         theta = 0.1
480         sigma = 0.1
481         L = 1
482         Q_0 = 1000
483         f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma
    **2) - (
484             Q_0
485             * (
486                 np.e ** (-(x**2) / theta**2)
487                 + np.e ** (-((x - L) ** 2) / theta**2)
488             )
489         )  # função f(x)
490         q = lambda x, y: 0  # função q(x)
491         for k in ks:
492             kx = lambda x, y: k  # função k(x)
493             alphas = sol_sistema_linear_tridiagonal(
494                 f, kx, q, n, n10
495             )  # solução do sistema
496
497             L = 1
498             h = L / (n + 1)
499             xi = [i * h for i in range(n + 2)]
500             u_barra_lista = []
501             for x in xi:
502                 u_barra_lista.append(u_barra(x, alphas, xi, h))
503                 print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi
    ,h):.22f}")
504
505             plt.plot(xi, u_barra_lista, label="$k= $" + str(k))
506         plt.grid()
507         plt.legend()
```

```
508         plt.ylabel("$T(x)$")
509         plt.xlabel("$x$")
510         plt.show()
511 
512         print("\nAnalisando para k(x)=ks se L/2-d<x<l/2+d e k(x)=ka
    caso contrario")
513         input("\nParametros: \nks=3.6, ka=60, L=1, d=L/10, q(x)=0, \
    nf(x)=Q(x)")
514 
515         n = 63
516         Q0 = 2000
517         theta = 0.1
518         sigma = 0.1
519         L = 1
520         Q_0 = 1000
521         f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma
    **2) - (
522             Q_0
523             * (
524                 np.e ** (-(x**2) / theta**2)
525                 + np.e ** (-((x - L) ** 2) / theta**2)
526             )
527         )  # função f(x)
528 
529         def k(x, y):  # função k(x)
530             L = 1
531             d = L / 10  # L/3
532             ka = 60
533             ks = 3.6
534             if L / 2 - d < x < L / 2 + d:
535                 return ks
536             else:
537                 return ka
538 
539         q = lambda x, y: 0  # função q(x)
540         alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  #
    solução do sistema
541 
542         L = 1
543         h = L / (n + 1)
544         xi = [i * h for i in range(n + 2)]
545         u_barra_lista = []
546         for x in xi:
547             u_barra_lista.append(u_barra(x, alphas, xi, h))
548             print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f}")
549 
550         plt.plot(xi, u_barra_lista, label="$d=L/10$, $k_a=60$")
551 
552         print("\nParametros: \nks=3.6, ka=60, L=1, d=L/3, q(x)=0, \
    nf(x)=Q(x)")
```

```
553         input("\nk(x)=ks se L/2-d<x<l/2+d e k(x)=ka caso contrario")
554
555         n = 63
556         Q0 = 2000
557         theta = 0.1
558         sigma = 0.1
559         L = 1
560         Q_0 = 1000
561         f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma
    **2) - (
562             Q_0
563             * (
564                 np.e ** (-(x**2) / theta**2)
565                 + np.e ** (-((x - L) ** 2) / theta**2)
566             )
567         )  # função f(x)  # função f(x)
568
569         def k(x, y):  # função k(x)
570             L = 1
571             d = L / 3  # L/3
572             ka = 60
573             ks = 3.6
574             if L / 2 - d < x < L / 2 + d:
575                 return ks
576             else:
577                 return ka
578
579         q = lambda x, y: 0  # função q(x)
580         alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  #
    solução do sistema
581
582         L = 1
583         h = L / (n + 1)
584         xi = [i * h for i in range(n + 2)]
585         u_barra_lista = []
586         for x in xi:
587             u_barra_lista.append(u_barra(x, alphas, xi, h))
588             print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f}")
589
590         plt.plot(xi, u_barra_lista, label="$d=L/3$, $k_a=60$")
591         plt.legend()
592         plt.grid()
593         plt.ylabel("$T(x)$")
594         plt.xlabel("$x$")
595         plt.show()
596
597         input("\nParametros: \nks=3.6, ka=429, L=1, d=L/10, q(x)=0,
    \nf(x)=Q(x)")
598
599         n = 63
```

```
600         Q0 = 2000
601         theta = 0.1
602         sigma = 0.1
603         L = 1
604         Q_0 = 1000
605         f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma
    **2) - (
606             Q_0
607             * (
608                 np.e ** (-(x**2) / theta**2)
609                 + np.e ** (-((x - L) ** 2) / theta**2)
610             )
611         )  # função f(x)  # função f(x)
612 
613         def k(x, y):  # função k(x)
614             L = 1
615             d = L / 10  # L/3
616             ka = 429
617             ks = 3.6
618             if L / 2 - d < x < L / 2 + d:
619                 return ks
620             else:
621                 return ka
622 
623         q = lambda x, y: 0  # função q(x)
624         alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  #
    solução do sistema
625 
626         L = 1
627         h = L / (n + 1)
628         xi = [i * h for i in range(n + 2)]
629         u_barra_lista = []
630         for x in xi:
631             u_barra_lista.append(u_barra(x, alphas, xi, h))
632             print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f}")
633 
634         plt.plot(xi, u_barra_lista, label="$d=L/10$, $k_a=429$")
635 
636         print("\nParametros: \nks=3.6, ka=429, L=1, d=L/3, q(x)=0, \
    nf(x)=Q(x)")
637         input("\nk(x)=ks se L/2-d<x<l/2+d e k(x)=ka caso contrario")
638 
639         n = 63
640         Q0 = 2000
641         theta = 0.1
642         sigma = 0.1
643         L = 1
644         Q_0 = 1000
645         f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma
    **2) - (
```

```
646             Q_0
647             * (
648                 np.e ** (-(x**2) / theta**2)
649                 + np.e ** (-((x - L) ** 2) / theta**2)
650             )
651         )  # função f(x)  # função f(x)
652 
653         def k(x, y):  # função k(x)
654             L = 1
655             d = L / 3  # L/3
656             ka = 429
657             ks = 3.6
658             if L / 2 - d < x < L / 2 + d:
659                 return ks
660             else:
661                 return ka
662 
663         q = lambda x, y: 0  # função q(x)
664         alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  # solução do sistema
665 
666         L = 1
667         h = L / (n + 1)
668         xi = [i * h for i in range(n + 2)]
669         u_barra_lista = []
670         for x in xi:
671             u_barra_lista.append(u_barra(x, alphas, xi, h))
672             print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h):.22f}")
673 
674         plt.plot(xi, u_barra_lista, label="$d=L/3$, $k_a=429$")
675         plt.legend()
676         plt.grid()
677         plt.ylabel("$T(x)$")
678         plt.xlabel("$x$")
679         plt.show()
680 
681         input("\nParametros: \nks=3.6, ka=0.03, L=1, d=L/10, q(x)=0, \nf(x)=Q(x)")
682 
683         n = 63
684         Q0 = 2000
685         theta = 0.1
686         sigma = 0.1
687         L = 1
688         Q_0 = 1000
689         f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma**2) - (
690             Q_0
691             * (
692                 np.e ** (-(x**2) / theta**2)
```

```
693                 + np.e ** (-((x - L) ** 2) / theta**2)
694             )
695         )  # função f(x)  # função f(x)
696 
697         def k(x, y):  # função k(x)
698             L = 1
699             d = L / 10  # L/3
700             ka = 0.03
701             ks = 3.6
702             if L / 2 - d < x < L / 2 + d:
703                 return ks
704             else:
705                 return ka
706 
707         q = lambda x, y: 0  # função q(x)
708         alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  #
    solução do sistema
709 
710         L = 1
711         h = L / (n + 1)
712         xi = [i * h for i in range(n + 2)]
713         u_barra_lista = []
714         for x in xi:
715             u_barra_lista.append(u_barra(x, alphas, xi, h))
716             print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f}")
717 
718         plt.plot(xi, u_barra_lista, label="$d=L/10$, $k_a=0.03$")
719 
720         print("\nParametros: \nks=3.6, ka=0.03, L=1, d=L/3, q(x)=0,
    \nf(x)=Q(x)")
721         input("\nk(x)=ks se L/2-d<x<l/2+d e k(x)=ka caso contrario")
722 
723         n = 63
724         Q0 = 2000
725         theta = 0.1
726         sigma = 0.1
727         L = 1
728         Q_0 = 1000
729         f = lambda x, y: Q0 * np.e ** (-((x - L / 2) ** 2) / sigma
    **2) - (
730             Q_0
731             * (
732                 np.e ** (-(x**2) / theta**2)
733                 + np.e ** (-((x - L) ** 2) / theta**2)
734             )
735         )  # função f(x)  # função f(x)
736 
737         def k(x, y):  # função k(x)
738             L = 1
739             d = L / 3  # L/3
```

```
740                 ka = 0.03
741                 ks = 3.6
742                 if (L / 2 - d) < x and x < (L / 2 + d):
743                     return ks
744                 else:
745                     return ka
746 
747         q = lambda x, y: 0  # função q(x)
748         alphas = sol_sistema_linear_tridiagonal(f, k, q, n, n10)  #
    solução do sistema
749 
750         L = 1
751         h = L / (n + 1)
752         xi = [i * h for i in range(n + 2)]
753         u_barra_lista = []
754         for x in xi:
755             u_barra_lista.append(u_barra(x, alphas, xi, h))
756             print(f"x = {x:.4f} ; u_barra = {u_barra(x,alphas,xi,h)
    :.22f}")
757 
758         plt.plot(xi, u_barra_lista, label="$d=L/3$, $k_a=0.03$")
759         plt.legend()
760         plt.grid()
761         plt.ylabel("$T(x)$")
762         plt.xlabel("$x$")
763         plt.show()
764 
765 
766 if __name__ == "__main__":
767     main()
```